YRKESHÖGSKOLAN NOVIA

Utveckling av en Windows Service för databassynkronisering med Microsoft Sync Framework

Christian Vik

Examensarbete för ingenjörsexamen (YH)

Utbildningsprogrammet för informationsteknik

Vasa 2011

EXAMENSARBETE
Författare: Christian Vik
Utbildningsprogram och ort: Informationsteknik, Vasa
Handledare: Kaj Wikman

Titel: *Utveckling av en Windows service för databassynkronisering med Microsoft Sync Framework*

Datum 31.03.2011 Sidantal 45

Abstrakt

Detta examensarbete utfördes åt Hogia Ferry Systems. Examensarbetet behandlar utvecklingen av ett synkroniseringsverktyg för databaserna som företagets produkter använder. Lösningen kommer att ersätta ett befintligt system för synkronisering.

Fokus för detta projekt ligger på utvecklingen av en synkroniseringstjänst för multipla typer av databaser. Databaserna som stöds är SQL Server, Informix samt Oracle. För utvecklingen av själva synkroniseringen har Microsoft Sync Framework använts. För att få konfigureringen och mappningen av databaserna dynamisk görs all konfigurering via XML-filer. För schemaläggning av synkroniseringen har Quartz.NET använts. Synkroniseringstjänsten har stöd för både schemalagd samt manuell synkronisering.

Projektet utvecklades på Microsoft .NET plattformen och utvecklingen skedde i Visual Studio 2010 med programmeringsspråket C#. Resultatet blev ett fungerande synkroniseringsverktyg implementerat som en Windows Service samt en klient skriven i WPF där kommunikationen med servicen sker via WCF.

Språk: svenska Nyckelord: databassynkronisering, Sync Framework, Quartz.NET, XML, WPF, WCF

OPINNÄYTETYÖ
Tekijä: Christian Vik
Koulutusohjelma ja paikkakunta: Tietotekniikka, Vaasa
Ohjaaja: Kaj Wikman

Nimike: *Windows service tietokannan synkronoinnin kehittäminen Microsoft Sync Frameworkilla*

Päivämäärä 31.03.2011 Sivumäärä 45

Tiivistelmä

Tämä opinnäytetyö tehtiin Hogia Ferry Systemsille. Tutkintotyö käsittelee synkronointityökalun kehittämisen tietokantaan, jota yrityksen tuotteet käyttävät. Ratkaisu tulee nykyisen ole synkronointijärjestelmän tilalle.

Tämä projekti kohdistaa synkronointipalvelun kehittämisen useimmille eri tietokannoille. Tietokannat, joita tuetaan ovat SQL Server, Informix sekä Oracle. Synkronointityön kehittämiseksi olen käyttänyt Microsoft Sync Frameworkia. Jotta saataisiin tietokantojen kokoonpano ja kartoitus dynaamiseksi, teen kaiken kokoonpanon XML-tiedostojien avulla. Synkronoinnin ajastamiseen olen käyttänyt Quartz.NETiä. Synkronointipalvelulla on tuki sekä ajastetulle että käsin tehtävälle synkronisoinnille.

Projekti kehitettiin Microsoft .NET perustalla ja kehitys tapahtui Visual Studio 2010:llä ohjelmointikielellä C#. Tuloksena saatiin toimiva synkronointityökalu toteutettuna kuten Windows Service sekä WPF:ssä kirjoitettu asiakasohjelma missä huollon kommunikaatio tapahtuu WCF:n kautta.

Kieli: ruotsi Avainsanat: tietokannan synkronointi, Sync Framework, Quartz.NET, XML, WPF, WCF

Arkistoidaan: Opinnäytetyö on saatavilla joko ammattikorkeakoulujen verkkokirjastossa Theseus.fi tai kirjastossa.

BACHELOR'S THESIS

Author: Christian Vik
Degree programme: Information Technology, Vaasa
Supervisor: Kaj Wikman

Title: *Development of a Windows Service for database synchronization using Microsoft Sync Framework*

Date 31.03.2011 Number of pages 45

Abstrakt

This Bachelor's thesis was made for Hogia Ferry Systems. The project deals with the development of a synchronization tool for the databases that the company's products use. The solution will replace an existing synchronization system.

The focus for this project is the development of a synchronization service for multiple types of databases. The supported databases are SQL Server, Informix and Oracle. Microsoft Sync Framework has been used for the development of the synchronization. To get the configuration and mapping dynamic, all the configuration are stored in XML files. Quartz.NET has been used for the scheduling of the synchronization. The synchronization service has support for both scheduled and manual synchronization.

The project was written with C# for the Microsoft.NET framework using Visual Studio 2010. The result is a working synchronization tool implemented as a Windows Service along with a Client written in WPF where the communication is done with WCF.

Language: Swedish Key words: database synchronization, Sync Framework, Quartz.NET, XML, WPF, WCF

Innehållsförteckning

Förkortningar och definitioner

Android	Android är ett operativsystem för bl.a. mobiltelefoner och utvecklas främst av Google och Open Handset Alliance.
Arbetsminne	Arbetsminne är det minne som används för att lagra de datorprogram som körs och dess data.
Assembly	En assembly i .NET är ett kompilerat kodbibliotek som används i den slutliga produkten. Dessa kan byggas som två olika typer: proces-assembly som är körbara program, EXE-filer, och library-assembly som är en DLL-fil.
BOOKIT	BOOKIT är ett system bestående av flera olika applikationer för färjebokningar. Mer om detta kan läsas i kap. 1.1.2.
BSD	BSD, Berkeley Software Distribution, utvecklades under 1970- och 1980-talet och är ett Unix-liknande operativsystem. Några operativsystem som används idag och som är baserade på detta är bl.a. FreeBSD, OpenBSD och Mac OS X.
COM	COM, Component Object Model, är en teknik som används för att kommunicera mellan olika objekt inom ett eller flera olika program. COM är plattforms- och språkoberoende.
COM+	COM+ är en vidareutveckling av COM.
Databas	I detta dokument behandlas relationsdatabaser. En relationsdatabas är en typ av databas som organiserar data i olika tabeller där kolumner i dessa tabeller kan ha en relation till andra tabeller. Dessa databaser behandlas mer i kap. 2.2.
Debugger	En debugger, översätts ibland till avlusare, är ett program som kan användas för att analysera ett program och dess minne under körning.
DLL	DLL, Dynamic Linked Library, är Microsofts implementation av delade programbibliotek. Detta betyder att en fil som innehåller specifika programfunktioner kan användas av flera olika program för att man t.ex. inte skall behöva inkludera gemensam kod i flera program.

GNU	GNU-projektet skapades av Richard Stallman 1984 med målet att utveckla ett fritt UNIX-liknande operativsystem. GNU är en rekursiv akronym som står för GNU is Not Unix, vilket på svenska blir GNU är inte Unix. GNU går under licensen GNU GPL
GNU GPL	GNU General Public License är en licens för fri programvara. Kod som är under GPL-licens är helt fri att använda och förbättra, men då man vidaredistribuerar programvaran måste man göra det under samma licens. Detta betyder att källkoden måste vara öppen för de program som är under GPL-licensen. Licensen i sin helhet hittas på adressen http://www.gnu.org/licenses/gpl.html. (23.03.2011)
GNU LGPL	GNU Lesser General Public License är en licens för fri programvara. Kortfattat får man använda kod från ett projekt som är under LGPL-licens i sitt eget projekt, men måste då använda samma licens för det egna projektet. Det är dock tillåtet att länka till ett externt bibliotek under LGPL-licens fastän man använder egen licens själv. Licensen i sin helhet hittas på webbadressen http://www.gnu.org/copyleft/lesser.html. (23.03.2011)
Implementera	Att implementera en metod är den praktiska tillämpningen av metoden.
Interface	Ett interface, eller gränssnitt, inom programmering är en typ av kontrakt. Ett objekt implementerar ett interface för att garantera att det stöder vissa operationer, på så vis kan man vara säker på att en viss metod finns på olika objekt om de bara implementerar det rätta interfacet.
iOS	iOS är ett operativsystem för mobiltelefoner och surfplattor och är utvecklat av Apple.
Java	Java är ett plattformsoberoende objektorienterat programmeringsspråk. Java nämns mer i kap. 2.7.
Klassbibliotek	Ett klassbibliotek är, i Windows miljö, oftast en DLL-fil. Ett klassbibliotek är en samling klasser och datastrukturer som kan användas till ett eller flera olika program.
Klient	En klient är oftast en typ av program som gör det möjligt att arbeta mot ett annat program som kan köras på någon annan dator.

Konstruktor	Inom programmering är en konstruktor en speciell funktion i ett objekt som alltid körs då objektet skapas.
Linux	Linux är en fri UNIX-liknande kärna för operativsystem som skapades av Linus Torvalds. I de flesta fall då man talar om Linux menar men egentligen GNU/Linux som är ett operativsystem som baserar sig på Linux-kärnan samt program från GNU-projektet.
Mac OS X	Mac OS X är ett operativsystem för Apples Macintosh-datorer.
ORM	ORM, Object/Relational Mapping, är ett system för att konvertera databastabeller till objekt inom objektorienterad programmering.
P2P	P2P, eller peer-to-peer, är ett nätverk som inte fungerar enligt server-klient-principen utan noderna i nätverket är ihopkopplade med andra noder.
Plugin	Ett plugin, kan även kallas insticksprogram eller modul, är ett program som inte körs som ett fristående program utan som en del av ett annat program. Plugin används oftast för att lägga till mer funktionalitet till ett program.
Proxy	En proxy är vanligtvis en server som ligger mellan en klient och en annan server. Proxyns uppgift är då att dirigera om trafiken mellan klienten och den andra servern.
Reflection	Reflection är en process där ett datorprogram kan granska och modifiera sin egen struktur och beteende under programmets körning.
RSS	RSS är en samling XML-baserade format som används för att publicera ofta uppdaterat material på webbsidor som t.ex. nyheter på nyhetssidor. Ett RSS-dokument innehåller oftast en full artikel eller en sammanfattning samt publiceringsdatum.
Server	Ett serverprogram är ett datorprogram som man kan ansluta till från ett klientprogram från antingen samma dator eller från någon annan dator eller mobil enhet över nätverk/internet.
Synkronisering	Synkronisering är då två eller flera objekt görs att t.ex. vara lika eller innehålla samma information vid en viss tidpunkt.
SQL	SQL (Structured Query Language) är ett standardiserat språk som används för relationsdatabaser för att hämta eller modifiera data.

TCP	TCP (Transmission Control Protocol) är ett dataöverföringsprotokoll som används t.ex. för kommunikation över internet.
UNIX	UNIX är en grupp operativsystem för datorer och började utvecklas 1969.
Windows	Windows är en serie operativsystem för datorer och utvecklas av Microsoft. Första versionen av Windows släpptes 1985
Windows Service	En Windows Service är ett program som ofta är utvecklat för att köras under lång tid och utan att behöva interaktion av användare. En Windows Service kan konfigureras att starta redan då Windows startar eller genom manuell start. Då en Windows Service är igång körs den i bakgrunden i systemet och om man behöver konfigurera eller arbeta mot den används oftast någon typ av klient, då servicen inte har något eget grafiskt gränssnitt.
WS	WS (Web Services), webbtjänster, är en metod för datorprogram att kommunicera med varandra över nätverk.
WSDL	WSDL (Web Services Description Language) är ett XML-baserat format för att beskriva funktionaliteten hos, och hur man anropar, en webbtjänst (WS).
XML	XML (eXtensible Markup Language) är ett utbyggbart språk som främst används för att utväxla data mellan olika system eller program. Datan i ett XML-dokument skickas som text som även går att läsas av människor.

1 Inledning

Examensarbetet utfördes åt företaget Hogia Ferry Systems, som är ett medelstort mjukvaruföretag i Korsholm. Examensarbetets uppgift gick ut på att planera och utveckla ett verktyg för att hålla databaser synkroniserade. De databaser detta främst gäller är databasen som används för deras huvudprodukt, kallad BOOKIT, men även övriga databaser kommer att gå att synkroniseras.

1.1 Företaget

1.1.1 Hogia

Programvaruföretaget Hogia grundades av Bert-Inge Hogsved 1980 i Stenungssund i Sverige och är ett helt familjeägt och självfinansierat företag. I dag består Hogia-gruppen av ett tjugotal företag som alla håller på med mjukvara i någon form. Varje enskilt bolag är komplett på det sättet att de ansvarar för allt från utveckling och försäljning till kundservice inom sitt verksamhetsområde. Bland de system som utvecklas av Hogia-företagen återfinns bl.a. system för kollektivtrafiken, färjetrafik, transport & lagerlogistik, terminalsystem och redovisningssystem.

1.1.2 Hogia Ferry Systems

Hogia Ferry Systems (HFS) är ett av företagen i Hogia-koncernen och finns i Korsholm, Finland. HFS grundades redan 1981 och hette då Consy. I samband med att man gick ihop med Hogia-gruppen byttes namnet till Hogia Ferry Systems. Idag har HFS ca 40 anställda. HFS har sedan 1987 utvecklat och sålt ett bokningssystem för färjetrafik kallat BOOKIT.

BOOKIT är ett system bestående av flera olika applikationer för färjebokningar. All information i BOOKIT:s olika applikationer sparas i en gemensam databas vilket ger den möjligheten att alla applikationer kan fritt dela information mellan varandra. Även om BOOKIT är en standardprodukt är den byggd för att vara modulär, skalbar och utökningsbar så att kunden får den funktionalitet de behöver.

BOOKIT består bl.a. av följande applikationer:

- BOOKIT Reservation
- BOOKIT Check-In
- BOOKIT Finance
- BOOKIT Reports

- BOOKIT Quick Sale
- Web Application
- BOOKIT SOA Plattform.

BOOKIT var från början skrivet i Informix 4GL, men 2001 började man utveckla en version i Microsoft .NET. I dag sker all nyutveckling på .NET-versionen och endast underhåll på den äldre 4GL-versionen för de kunder som fortfarande inte uppgraderat till .NET-versionen.

1.2 Projektbeskrivning

Projektet går ut på att ersätta det gamla synkroniseringsverktyget med en snabbare och mer tillförlitligt lösning. Den nya synkroniseringslösningen kommer att implementeras som en Windows-service. Servicen kommer att göras så att den kan köras på i princip vilken dator som helst i nätverket, bara den har tillgång till de databaser som ska synkroniseras.

För att kunna konfigurera synkroniseringsservicen, t.ex. att ställa in vilken/vilka databaser som ska synkroniseras och när, kommer också en klient att utvecklas. Klienten och servern ska dessutom kunna kommunicera med varandra både såväl lokalt som över nätverket.

1.2.1 Existerande lösning

Den existerande lösningen som används för synkronisering av databaserna är ett plugin för TaskScheduler som är ett eget utvecklat program som används för att köra schemalagda aktiviteter. Den plugin för TaskScheduler som används för synkroniseringen är begränsad till att synkronisera enbart en tabell åt gången, vilket betyder att för att kunna synkronisera hela databasen måste flera instanser av samma plugin köras. Detta medför problem i och med att flera, eller i värsta fall alla, tabeller kan börja synkroniseras under samma tillfälle. Detta kan sänka prestandan betydligt och det betyder också att man inte kan ha några relationer mellan tabellerna i databasen som det ska synkroniseras till.

Själva synkroniseringen fungerar så att pluginen kör ett SQL-kommando mot BOOKIT:s databas för att se vilka rader som ändrats sedan senaste synkroniseringen. Data hämtas sedan från databasen i batchar med ett specificerat antal rader per batch och kopieras sedan in i CRM-databasen. CRM-databasen används av kunderna t.ex. för att ta ut tyngre statistik rapporter som annars skulle påverka prestandan negativt för BOOKIT.

1.2.2 Begränsningar & krav

Ett av kraven på den nya synkroniseringen är att den måste stöda tre typer av databaser, MS SQL Server, Oracle och Informix. Detta medförde också att de ramverk och klassbibliotek som används måste stöda dessa typer av databaser. Det andra kravet var att synkroniseringen skulle gå att konfigurera via något verktyg så att man inte behöver ändra i programmets kod varje gång databasen ändras. Utöver detta är det enda övriga kravet att synkroniseringen ska köras under Windows Vista eller nyare.

2 Programvara och utvecklingsverktyg

Det utvecklingsverktyg som använts för all programmering och utveckling av programmet är Microsofts Visual Studio 2010 och det programmeringsspråk som främst använts är C#, Visual Basic.NET har också använts i viss mån men endast för de projekt som refereras från BOOKIT. C# valdes som språk både p.g.a. egen erfarenhet, men också därför att största delen av den dokumentation och exempelkod som finns över de använda ramverken och komponenterna är gjorda i C#.

Några alternativ till Visual Studio är MonoDevelop och SharpDevelop. MonoDevelop är en fri utvecklingsmiljö som stöder både Mono och Microsoft.NET samt kan köras på Linux, Windows och Mac OS X. MonoDevelop har även stöd för C# och Visual Basic.NET. SharpDevelop är, liksom MonoDevelop, en fri utvecklingsmiljö med stöd för bl.a. C# och Visual Basic.NET.

Valet föll ändå på att använda Visual Studio för utvecklingen, dels för att det redan används för övriga projekt och vissa av dessa projekt kommer att refereras av synkroniseringsprojekten och dels för att många av de program och tjänster som kommer att användas är från Microsoft och integrationen med Visual Studio är bättre.

De databaser som synkningen har blivit implementerad och testad med är MS SQL Server 2008 R2 Standard, Oracle 11g och Informix 10 och 11.5. Dessa databaser var inte valbara eftersom det är de databaser som redan stöds och används av företagets andra produkter.

2.1 Microsoft Visual Studio 2010 Professional

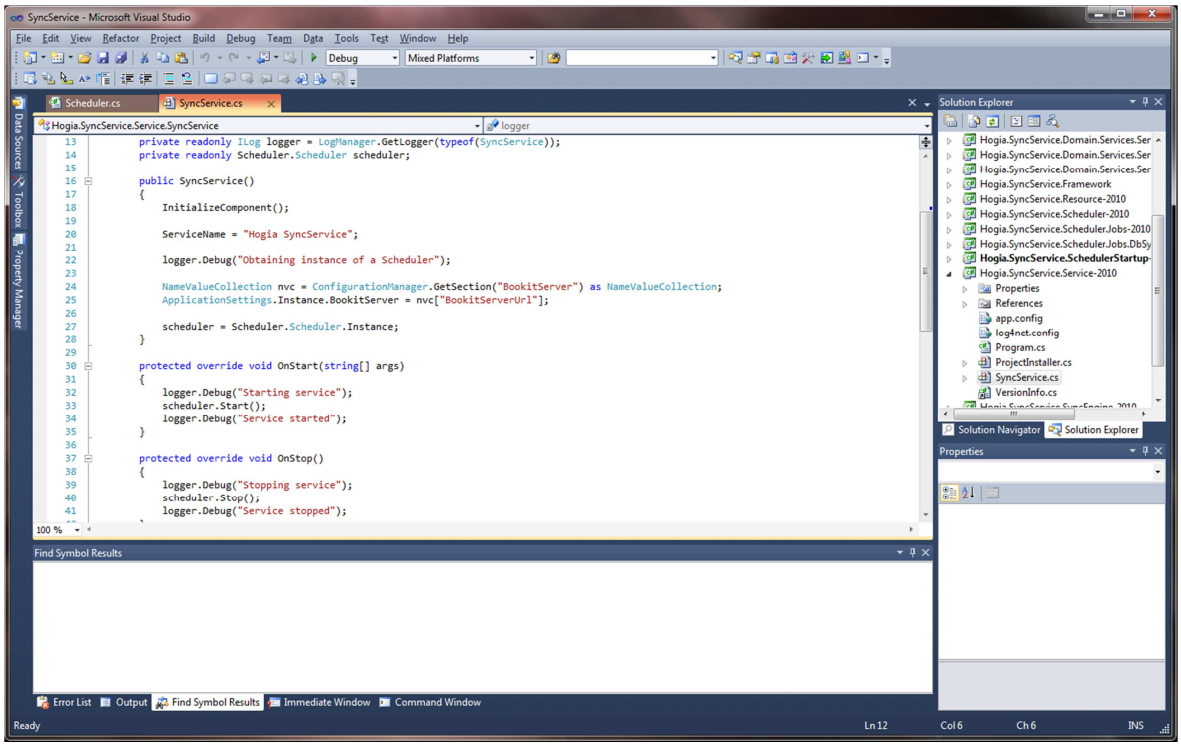
Valet av Visual Studio 2010 framför Visual Studio 2008 var på grund av att 2010 har stöd för .NET 4, vilket behövs för att kunna använda Entity Framework 4.

Microsoft Visual Studio 2010 är en integrerad utvecklingsmiljö från Microsoft som används främst för programutveckling. Visual Studion kan användas för utveckling av både konsolapplikationer och grafiska applikationer, som t.ex. Windows-program och webbanpassade applikationer och system. Visual studion har stöd för bland annat följande programmeringsspråk: C/C++, VB.NET (Visual Basic .NET), C# (Visual C#) och F#. /16/

Kod-editorn i Visual Studio har support för bl.a. IntelliSense som är Microsofts implementation av kod auto-komplettering, kodomstrukturering (eng. code refactoring) och syntaxmarkering (eng. syntax highlighting). Den inbyggda debuggern har bl.a. stöd för program skrivna både .NET och i native windows-språk som t.ex. C++. Debuggern har också stöd för s.k. breakpoints, som gör att programmet kan stanna och vänta på en given kodrad så

att man kan undersöka hur olika variabler påverkats och vid behov ändra koden utan att behöva starta om programmet som testas. /8/

I övrigt har Visual Studio stöd för integration av tredjeparts-komponenter och ramverk. Beroende på vilken typ av komponent man installerar läggs även extra funktionalitet in i Visual Studio, t.ex. rapportverktyg brukar komma med en designer för att skapa layouten till rapporten. Visual studio 2010 har även stöd för s.k. extensions som gör att man kan installera ytterligare funktionalitet direkt från programmet.



Figur 1. Visual Studio 2010 kodeditor.

2.2 Databaserna

Synkroniseringstjänsten kommer att stöda tre olika databaser, Microsoft SQL Server, Informix och Oracle. Orsaken till det är att våra nuvarande produkter redan stöder dessa, så synkroniseringen måste också göra det. Alla dessa databaser är av typen relationsdatabaser och använder sig av SQL-kommandot. Ett SQL-kommando är en fråga eller kommando till databasen för att t.ex. hämta eller manipulera data. SQL är standardiserat, men alla dessa databaser har lite olika variationer av språket som måste tas i beaktande då synkroniseringstjänsten skrivs.

Här följer exempel på fyra av de kanske mest använda SQL-kommandona:

```
SELECT column1, column2 FROM table1 WHERE column3 = “value”
```

Figur 2. SQL Select-kommando

Detta kommando hämtar kolumnerna ”column1” och ”column2” från tabellen ”table1” där kolumnen ”column3” har värdet value.

```
DELETE FROM table1 WHERE column3 = “value”
```

Figur 3. SQL Delete-kommando

Detta kommando tar bort alla rader från tabellen ”table1” där kolumnen ”column3” har värdet value.

```
INSERT INTO table1 (column1, column2, column3) VALUES (“value1”, “value2”, “value”)
```

Figur 4. SQL Insert-kommando

Detta kommando lägger in en rad i tabellen ”table1” där kolumnen ”column1” får värdet ”value1”, ”column2” värdet ”value2” och ”column3” värdet ”value3”.

```
UPDATE table1 SET column1=”value1”, column2=”value” WHERE column3 = “value”
```

Figur 5. SQL Update kommando

Detta kommando uppdaterar de rader i tabellen ”table1” där kolumnen ”column3” har värdet ”value”. I de uppdaterade raderna kommer värdet i kolumn ”column1” att få värdet ”value1” och kolumn ”column2” värdet ”value2”.

En relationsdatabas är en typ av databas där informationen är organiserad i tabeller, eventuellt med relationer inom en tabell eller med andra tabeller. En rad i en tabell kan ofta motsvara någon typ av objekt, t.ex. en person, ett företag eller en beställningsorder.

Ofta används åtminstone en restriktion i en tabell, den s.k. primärnyckeln som identifierar unika rader i en tabell. Om man har en tabell över t.ex. beställningsorders så kunde ordernummern vara primärnyckel.

Främmande nycklar är en hänvisning till en nyckel i en annan tabell, t.ex. om man har en tabell för personer och en tabell för beställningar så kan det finnas en främmande nyckel mellan dessa två tabeller som kopplar ihop beställningarna med den person som gjorde dem.

Alternativ till relationsmodellen som används av relationsdatabaser är:

- Flat modell – Består av en tvådimensionell tabell där all data lagras.
- Hierarkisk modell – I en hierarkisk modell är data organiserad i en trädstruktur, t.ex. Parent-Child relation där en parent kan ha flera child noder, men en child nod kan bara ha en parent. Förutom databaser följer t.ex. XML denna typ.

- Nätverksmodell – Ganska lik den hierarkiska modellen förutom att en child kan relatera till flera parents.
- Dimensionsmodell – Dimensionsmodellen är en special anpassning av relationsmodellen som används för t.ex. summeringar i OLAP. I dimensionsmodellen finns en fakta tabell som beskrivs av s.k. ”dimensions” och ”measures” tabeller. En dimension ger ett sammanhang över fakta tabellen medan en measure är en storhet som beskriver faktatabellen, t.ex. antal eller intäkter.
- Objekt-relationsmodell – Objekt-relationsmodellen är lika relationsmodellen men med en objektorienterad databasmodell.

2.2.1 Microsoft SQL Server 2008 R2

Microsoft SQL Server är en databashanterare från Microsoft. SQL Server är av typen relationsdatabas. Som frågespråk används SQL, eller närmare bestämt T-SQL som står för Transact-SQL.

Första versionen av Microsoft SQL Server, 1.0, släpptes 1989 och den senaste versionen (23.03.2011) är 10.5 och går under namnet SQL Server 2008 R2. Till skillnad från flera av de andra stora databaserna kan Microsoft SQL Server numera enbart köras under operativsystemet Windows.

2.2.2 IBM Informix

IBM Informix är en produktfamilj av relationsdatabaser (eng. Relational Database Management System) (RDBMS). Informix-projektet skapades av företaget Relational Database Systems 1981. Företaget bytte sedan namn till Informix Software 1986. 2001 köpte IBM upp Informix Softwares databasteknologi och allt däromkring samt dess kundbas. /3/

IBM Informix är designat från grunden för enkel skalbar administration och många funktioner som tillåter Informix att ”försvinna” in i en applikation. I Informix har man också satsat på tillförlitlighet och flexibel replikering samt redundans alternativ för kontinuitet i kundens verksamhet. Informix kan köras på följande plattformar: Windows, Mac OS X, Linux, BSD, UNIX och z/OS./4/

2.2.3 Oracle

Oracle, eller Oracle Database, är liksom SQL Server och Informix också av typen relationsdatabas, eller närmare bestämt ORDBMS som står för Object-Relational Database Management System. ORDBMS liknar en traditionell relationsdatabas men med en objektorienterad databasmodell där objekt, klasser och arv stöds direkt i databasscheman och i frågespråket. Oracle använder SQL som frågespråk.

Oracle produceras och marknadsförs av Oracle Corporation och grundades 1979. Oracle kan köras på följande plattformar: Windows, Mac OS X, Linux, UNIX och z/OS. /13/

2.3 Microsoft .NET Framework

Microsoft .NET Framework är ett programvaruramverk för Microsofts Windows operativ system. .NET-ramverket har kommit med som standard sedan Windows Server 2003, vilket betyder att alla Windows Server-versioner sedan 2003 samt Windows Vista och Windows 7 har .NET ramverket färdigt installerat vid leverans. Microsoft .NET har stöd för flera olika programmeringsspråk, inklusive C# och VB.NET, och stöder också att ett program kan skrivas med flera olika programmeringsspråk. Basklassbiblioteket ger sådan funktionalitet som användargränssnitt, dataåtkomst, databasanslutningar, kryptografi, webapplikationer och nätverkskommunikation. Detta klassbibliotek används då av programmerare tillsammans med eventuella andra klassbibliotek samt egen utvecklad kod för att skapa de önskade funktionerna/applikationerna.

Program som är skrivna i .NET-ramverket körs i en mjukvarumiljö under namnet Common Language Runtime (CLR). CLR fungerar som en virtuell maskin så att programmerarna inte behöver ta i beaktande de specifika hårdvarukonfigurationerna. /11/

Ett alternativ till Microsoft.NET är Mono. Mono är ett gratis öppen källkodsbaserat projekt som leds av Novell för att skapa en Ecma-standard .NET-kompatibel uppsättning verktyg, bl.a. en C# kompilerare och CLR. Till skillnad från Microsoft.NET, som bara fungerar i Windows, fungerar Mono på bl.a. Android, BSD, iOS, Linux, Mac OS X och Windows. Mono har full support för C# 4.0, men flera av de teknologier som finns i Microsoft.NET saknas ännu i Mono, bl.a. WPF, Entity Framework, Workflow Foundation och WCF.

2.4 Microsoft Sync Framework

Microsoft Sync Framework är en omfattande synkroniseringsplattform med möjligheter för samarbete och off-line-läge för applikationer och tjänster mellan datorer och mobila enheter. En av huvudfunktionerna i Sync Framework är att man kan skapa egna providers, vilket gör det möjligt att synkronisera i princip vilken typ av data som helst. En provider för en viss typ av datakälla gör det möjligt att synkronisera denna typ. Providers som finns med i Sync Framework är:

- Databassynkroniseringsprovider för databaser med ADO.NET drivrutiner.
- Filsynkroniseringsprovider för filer och kataloger.
- Webbsynkroniseringsprovider för t.ex. RSS.

Sync Framework kan användas för att arbeta med data utan att behöva vara uppkopplad mot datakällan för att sedan då man är färdig bara skicka ändringarna. Då en klient skickat in ändringar finns också möjligheter att kunna meddela övriga klienter att ändringar skett och på så vis hålla dem synkroniserade, s.k. publish/subscribe sync, samt P2P-synkronisering. /7/

Sync Framework har även inbyggd funktionalitet för upptäckande av konflikter, d.v.s. när den data en klient ändrat redan blivit ändrad av en annan klient sedan den senaste synkroniseringen, och kan då markera dem för manuell bearbetning eller använda förhandsdefinierade metoder.

För att synkronisera data mellan t.ex. två databaser behövs inte direkt tillgång till databaserna, utan Sync Framework kan också använda bl.a. web services och WCF bara det finns en proxy för att abstrahera bort datakällan.

Klienterna delas upp i tre olika kategorier:

- Full deltagare (Full Participant)
- Partiell deltagare (Partial Participant)
- Enkel deltagare (Simple Participant)

En full deltagare är sådana enheter där utvecklarna kan installera programvara direkt på enheten, t.ex. en dator eller smarttelefon är sådana enheter.

En partiell deltagare är en sådan enhet som kan lagra data men där man inte kan köra egna program. Exempel på sådana enheter är t.ex. masslagringsenheter.

En enkel deltagare är sådana som bara kan ge information då man frågar efter den. Dessa enheter kan inte lagra eller ändra informationen. RSS är ett exempel på en sådan.

2.5 Entity Framework 4

ADO.NET Entity Framework (EF) är ett ORM ramverk för .NET-ramverket. Syftet med EF är att abstrahera bort relationsschemamodellen som används i en relationsdatabas till objekt/klasser i .NET. Detta medför då att man i programkoden inte behöver känna till den exakta databasstrukturen samt att det blir lättare att anpassa programmet för ändringar i databasen.

Entity Framework är databasoberoende och använder sig av ADO.NET för kommunikation med databaser. Detta betyder att EF skall vara kompatibel med en databas om det finns en

ADO.NET Provider för den. EF använder sig i grunden av SQL-kommandon för att kommunicera med databasen, men dessa kommandon exponeras aldrig mot användaren utan denne använder sig av de metoder EF erbjuder för att arbeta mot databasen. Man kan däremot inte editera databasstrukturen via EF.

En provider för EF finns åtminstone för följande databaser:

- Sql Server
- MySQL
- Oracle
- SQLite
- PostgreSQL
- DB2
- Informix

Första versionen av EF (EFv1) inkluderades med .NET 3.5 Service Pack 1 samt Visual Studio 2008 Service Pack 1. Den andra versionen av EF, kallas Entity Framework 4.0, inkluderades som en del av .NET 4.0.

Det finns också flera alternativ till Entity Framework, här nedan följer några:

- NHibernate
- SubSonic
- Mindscape's LightSpeed

NHibernate är liksom Entity Framework en ORM-lösning för Microsoft .NET-ramverket. NHibernate är en portad version av Hibernate som är skriven i Java och är i nuvarande version, 3.0.0, en fri och öppen källkodsprojekt som går under licensen GNU LGPL. Från och med version 3.0.0 har NHibernate stöd för Microsoft .NET 3.5. /12/

SubSonic har mindre funktionalitet än NHibernate men fokuserar i stället på att vara mer flexibelt och snabbare att använda. SubSonic fungerar dessutom mot de flesta databaser./1/

LightSpeed, till skillnad från övriga nämnda alternativ, är inte en gratis produkt. Det finns en fri Express-version, men den är begränsad till 8 modeller/klasser. Förutom stöd för Microsoft.NET har LightSpeed även stöd för Mono samt flera olika databaser. /9/

2.6 Quartz.net

Quartz.NET är ett jobbschemaläggningssystem med öppen källkod. Quartz.NET är en portad version av ett jobbschemaläggningsramverk skrivet i programmeringsspråket Java. Quartz.NET kan köras som en del av ett annat program eller som ett eget program. Det kan också köras som ett kluster av flera program för belastningsbalans och/eller redundans funktionalitet.

Quartz.NET:s jobbschemaläggning har stöd för s.k. triggers som bl.a. anger vilka tidpunkter ett jobb ska köras. En trigger behöver också start- och eventuellt stoppdatum för när den skall vara aktiv. En trigger kan skapas med kombinationer av dessa direktiv:

- på en given tidpunkt på dagen
- på givna veckodagar
- på givna dagar i månaden eller året
- upprepas ett specificerat eller oändligt antal gånger
- upprepas till en specificerad tid
- upprepas med ett intervall.

Nuvarande version av Quartz.NET, 1.0.3, har stöd för .NET 1.1 till 3.5, men fastän det inte har officiellt stöd för .NET 4 ännu så har det fungerat utan problem i detta projekt. /14/

2.7 C#

Visual C# är Microsofts implementering av programmeringsspråket C#. C# är ett programmeringsspråk designat för att utveckla variationer av program för .NET-ramverket. Visual Studio har fullt stöd för C# med bl.a. kodeditorn, kompilerare, projektmallar, designer och en kraftfull samt lättanvänd debugger. /15/

Eftersom C# har en hel del influenser från C++ liknar dessa två språk varandra i viss mån. Då även Java också bl.a. baserar sig på C++, även om C# och Java tagit sina egna spår, går det relativt enkelt att förstå program skrivna i de andra språken och att porta ett program mellan de olika språken. Utöver dessa tre vanligt förekommande språk finns även Visual Basic.NET (VB.NET) som, liksom C#, baserar sig på .NET-ramverket. Funktionsmässigt är det väldigt lika C# då de funktioner som finns i .NET-ramverket är samma oberoende språk, men syntaxmässigt är VB.NET väldigt olik de olika C-deriverade språken.

Nedan följer det klassiska Hello World-programmet i en version för vardera av de fyra språken. Hello World-programmets enda syfte är att skriva ut texten ”Hello, World!” på skärmen då det körs.

```
#include <iostream>

void main()
{
        std::cout << "Hello, world!";
}
```

Figur 6. Hello World i C++.

```
using System;
class ExampleClass
{
    static void Main()
    {
        Console.WriteLine("Hello, world!");
    }
}
```

Figur 7. Hello World i C#.

```
public class HelloWorld
{
      public static void main(String[] args)
      {
            System.out.println("Hello, World!");
      }
}
```

Figur 8. Hello World i Java.

```
Imports System

Module Module1

    Sub Main()
        Console.WriteLine("Hello, world!")
    End Sub

End Module
```

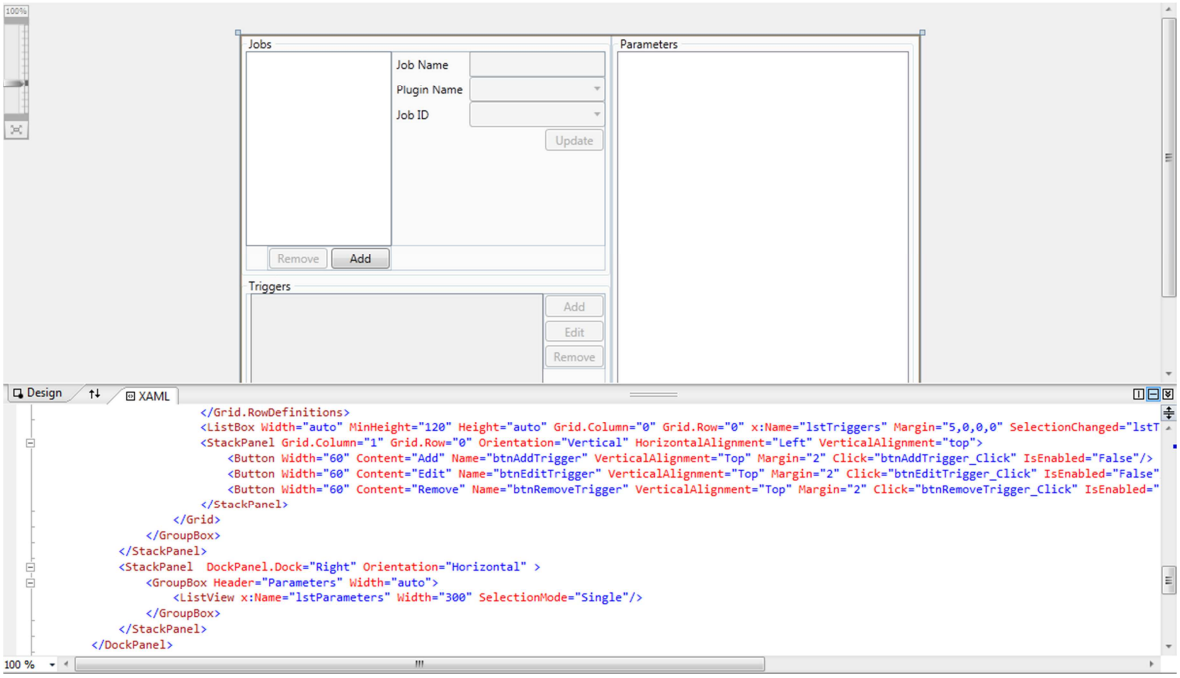
Figur 9. Hello World i Visual Basic.NET.

2.8 WPF – Windows Presentation Foundation

WPF är ett grafiskt system för att skapa och rendera grafiska användargränssnitt för Microsoft Windows-baserade applikationer. WPF har funnits med i .NET-ramverket sedan version 3.0 och har kommit som standard i Windows Vista, Windows 7 och Windows Server 2008. Det finns även tilläggskomponenter för Windows XP SP2/SP3 och Windows Server 2003 för att installera stöd för WPF.

WPF är främst menat för programutvecklare och grafiker som vill skapa moderna och användarvänliga användargränssnitt utan att behöva lära sig många olika teknologier. Några av funktionerna i WPF är:

- Bred integration – WPF har inbyggt stöd för 3D (via DirectX), video, ljud, text hantering och vanliga 2-dimensionella grafiska kontroller som alla kan användas med samma programmeringsmodell. Med tidigare alternativ har man behövt lära sig de olika delarna separat och försöka få dem integrerade med varandra så gott det gått.
- Upplösningsoberoende – Program skrivna i WPF har den möjligheten att de kan förminskas och förstoras, både hela fönster eller vissa komponenter, oberoende av upplösningen på skärmen utan att för den skull se förstörda ut. Detta är till stor del möjligt p.g.a. att WPF använder vektorgrafik.
- Hårdvaruacceleration – Eftersom WPF är byggt ovanpå Direct3D så finns det fullt stöd för hårdvaruacceleration i WPF. Allt innehåll i en WPF-applikation, oberoende av om det är 2D eller 3D, konverteras till 3D-trianglar samt andra Direct3D-objekt som sedan renderas av hårdvaran. Om hårdvaran inte stöder 3D faller renderingen automatiskt över till mjukvarurendering som dock inte är lika snabb som hårdvarurenderingen då den belastar datorns processor mer.
- Deklarativ programmering – I WPF är användargränssnittet separerat från programlogiken. Användargränssnittet byggs upp av XAML, Extensible Application Markup Language, -filer. WPF och XAML kan liknas vid hur man använder HTML för att bygga upp utseendet på en webbsida. Genom att separera programlogiken och användargränssnittet finns den möjligheten att grafiker kan lättare och snabbare skapa eller ändra utseendet på ett program utan att nödvändigtvis kunna programmera.
- Enkel distribuering – WPF, liksom Windows Forms, har stöd för olika former av distribuering. Det är möjligt att använda bl.a. Windows Installer eller ClickOnce för vanliga program eller att köra programmet i en webbläsare. /10/

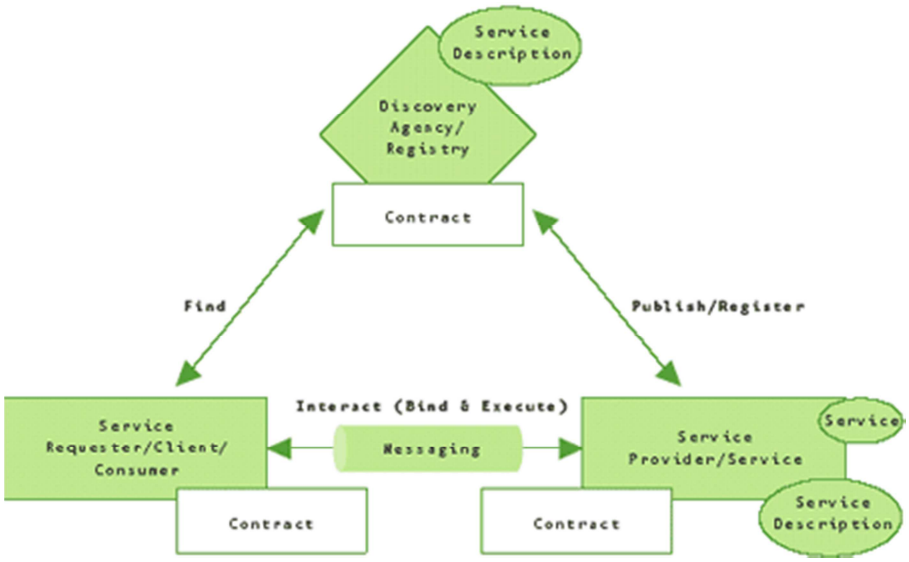


Figur 10. Visual Studio 2010 WPF designer med XAML-editor.

2.9 WCF – Windows Communication Foundation

WCF, Windows Communication Foundation eller Indigo, som dess kodnamn var under utvecklingen, är en teknologi som gör det möjligt för olika program eller delar av program eller system att kommunicera med varandra. Det är m.a.o. ett ramverk för att bygga s.k. serviceorienterade applikationer. Några av de funktioner som finns i WCF:

- SOA – Service-Oriented Architecture innebär att distribuerade informationssystem organiseras som kommunicerande tjänster. SOA kan bestå av olika entiteter enligt figuren nedan. /6/



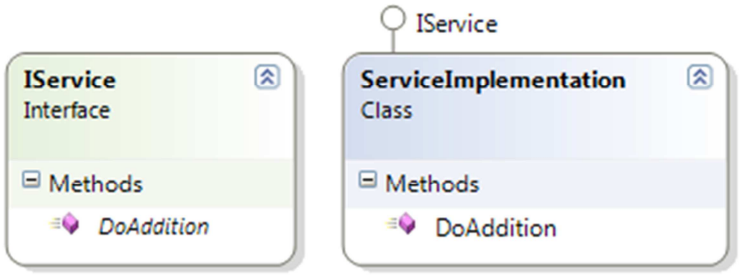
Figur 11. Schema över en SOA arkitektur.

Förklaring av figur 11:

- Service Consumer är t.ex. en applikation som behöver en service. Det är denna entitet som söker upp servicen från serviceregistret, tar upp en förbindelse och kör servicens funktion. Service Consumern kör servicen genom att sända ett anrop enligt kontraktet.
- Service Providern är en adresserbar entitet i nätverket som accepterar och kör anrop som kommer från Consumern. Providern publicerar sitt kontrakt i registret så att Consumern kan få tag i det.
- Service Registry är en entitet som accepterar och lagrar kontrakt från service providers och ger ut dessa till service consumers.
- Ett Contract är en specifikation över hur en consumer skall kommunicera med providern.

- Serviceorienterat – En serviceorienterad arkitektur har den fördelen att servicen är löst kopplad istället för hårdkodad mellan olika applikationer. Att vara löst kopplade betyder att vilken klient på vilken plattform som helst kan ansluta till servicen så länge de väsentliga kontrakten uppfylls.
- Interoperabilitet – WCF kan samarbeta med flera andra olika teknologier, som t.ex. Web Service Protocol, COM+, COM och .NET remoting.
- Servicemetadata – WCF har stöd för att publicera servicemetadata i industristandarder som WSDL, XML och WS. Metadatan kan sedan användas för att automatiskt generera och konfigurera klienter för att kunna ansluta till WCF-servicen.
- Data contract – För att hantera den data som skickas över WCF kan man skapa klasser som representerar den. WCF genererar sedan automatiskt det metadata som klienten behöver.
- Säkerhet – Meddelanden som går över WCF kan krypteras och man kan också göra så att en användare måste identifiera sig före den kan ta emot meddelanden. Säkerheten kan implementeras med standarder som t.ex. SSL eller WS-SecureConversation. /5/

För att skapa en WCF-server och -klient behövs relativt lite kod. Här nedan följer ett klassdiagram samt exempel på en enkel implementation för en server/klient.



Figur 12. Klassdiagram för ett WCF-kontrakt samt implementationen.

Först skapas service kontraktet som WCF-servicen kommer att implementera. Detta kontrakt exponeras senare till klienten så att den får informationen den behöver för att kunna anropa servicen.

```
using System.ServiceModel;
namespace SimpleWCF
{
    [ServiceContract]
    interface IService
    {
        [OperationContract]
        int DoAddition(int Value1, int Value2);
    }
}
```

Figur 13. Definiering av ett WCF-kontrakt.

Efter att kontraktet är klart implementeras servicen. Service-klassen implementerar interfacet (kontraktet) ovan. Funktionen DoAddition tar två parametrar och returnerar summan av dem.

```
using System.ServiceModel;
namespace SimpleWCF
{
    [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall)]
    public class Service : IService
    {
        public int DoAddition(int Value1, int Value2)
        {
            return Value1 + Value2;
        }
    }
}
```

Figur 14. Implementation av WCF-service.

Med koden nedan startas en WCF-server.

```
ServiceHost svh = new ServiceHost(typeof(Service));
  svh.AddServiceEndpoint(typeof(IService),
  new NetTcpBinding(),
  "net.tcp://localhost:8000");
svh.Open();
```

Figur 15. Instansiering av en WCF-service.

Med koden nedan ansluter man till servern som startades tidigare och anropar funktionen DoAddition med parametrarna 5 och 10. Efter att server tagit emot anropet, behandlat det och returnerat ett värde kommer en meddelanderuta att visas med talet 15.

```
ChannelFactory<IService> cf = new ChannelFactory<IService>(
  new NetTcpBinding(),
  "net.tcp://localhost:8000");
IService s = cf.CreateChannel();
int response = s.DoAddition(5, 10);
(s as ICommunicationObject).Close();
MessageBox.Show(response.ToString());
```

Figur 16. Ansluta till en WCF-Service.

3 Utförande

3.1 Inledning

Synkroniseringsverktyget kommer att utvecklas med något existerande ramverk för hantering av dataöverföring mellan databaserna. Valet står mellan Sync Framework och Entity Framework, så förrän utvecklingen kan påbörjas av själva verktyget måste en utvärdering ske av dessa två ramverk. Verktyget kommer sen att implementeras som en Windows-service med funktionalitet för schemalagda synkroniseringar och loggning. Utöver Windows-servicen kommer även en klient att skapas för att kunna köra manuella synkningar och för att konfigurera när synkningarna skall köras.

3.2 Sync Framework vs Entity Framework

Före utvecklingen av synkroniseringsverktyget, härefter SyncService, kunde påbörjas var man tvungen att ta ett beslut på vilket ramverk som skulle användas för hanteringen och överföringen av data. Som alternativ fanns Microsoft Sync Framework och Entity Framework. För att ta det beslutet beslöts att man utvecklar ett enkelt program i två versioner som använder de respektive två ramverken. Programmets enda syfte var att göra en envägssynkronisering av data från två tabeller mellan två databaser där databasschemat var identiskt förutom att tabell- och kolumnnamn var översatta till engelska i klientdatabasen.

Programmen implementerades så simpelt som möjligt, d.v.s. ett enda projekt som använde sig av de två ramverken och enbart den nödvändiga koden för att möjliggöra en envägssynkronisering av data från en databas till en annan. Som programmeringsspråk valdes C# och utvecklingen skedde i Visual Studio 2010. Databasen som användes i testet var MS SQL Server 2008 R2 Standard.

I de tester som kördes med dessa två program visade det sig att prestandamässigt var de två ramverken i princip lika effektiva i denna enklaste variant. I samtliga tester där mängden data varierade mellan 15.000 och 50.000 rader var Entity Framework 1–2 sekunder snabbbare. På grund av den lilla tidsskillnaden valde man att avgöra beslutet på mängden arbete att implementera en fullt fungerande lösning och på de båda ramverkens för- och nackdelar.

MS Sync Framework

Fördelar:

- Inbyggd funktion för hantering av s.k. ”*anchors*” som bestämmer vilket intervall av data som ska synkroniseras och vad som redan blivit synkroniserat.

- Har stöd för att använda direkta SQL-kommandon för att hämta och spara data.
- Har stöd för s.k. ”*Batching*”, vilket gör det möjligt att dela upp det data som ska synkroniseras i mindre delar för att undvika att datorns arbetsminne inte fylls upp under synkronisering av större mängder data.
- Går bra att använda diverse olika datakällor eftersom man kan implementera egna *DataProviders* för att ansluta till dessa olika källor.

Nackdelar:

- Väldigt komplicerat om databaserna som ska synkroniseras inte har samma schema.
- Delvis bristfällig dokumentation och väldigt få fungerande exempel.
- Tar betydligt längre att komma igång med än Entity Framework

Entity Framework 4

Fördelar:

- Enkelt att komma igång med.
- Väldigt flexibelt då man själv kan implementera hela överföringen av data.
- Går bra att synkronisera data mellan databaser som har helt olika scheman då man själv kan transformera datan.

Nackdelar:

- Eftersom Entity Framework inte har någon färdig implementation av överföringen av data från en databas till en annan måste man implementera den själv, detta gäller också funktioner som t.ex. anchors och batchning som finns i Sync Framework.
- Fungerar inte bra då databaslagret skall vara transparent, d.v.s. att programmet ska fungera lika oberoende av vilken av de tre databaserna som används.

På grund av att datan som skall synkroniseras skall vara identisk mellan de två databaserna och då det behöver stöda flera typer av databaser så beslöt man att MS Sync Framework är bättre lämpad för det här projektet. Då MS Sync Framework har inbyggt stöd för anchors och batchning av data så behöver man inte heller implementera detta, så som man hade blivit tvungen att göra med Entity Framework. Om uppgiften däremot hade krävt att datan skulle transformeras eller behandlas på något vis, t.ex. för att föra över data till en databas med ett annorlunda schema, så skulle Entity Framework ha valts.

3.3 Synkroniseringstjänsten

3.3.1 Implementation av Windows servicen

Servicedelen av SyncService implementeras som en enkel Windows-service, d.v.s. en klass som ärver basklassen ServiceBase som finns implementerad i Microsoft .NET under System. Serviceproces- biblioteket. Denna klass åter implementerar den funktionalitet som sker då servicen startas, stoppas och pausas.

```
static class Program
{
    static void Main()
    {
        ServiceBase[] ServicesToRun;
        ServicesToRun = new ServiceBase[] { new SyncService() };
        ServiceBase.Run(ServicesToRun);
    }
}

public partial class SyncService : ServiceBase
{
    private readonly Scheduler.Scheduler scheduler;

    public SyncService()
    {
        InitializeComponent();
        ServiceName = "Hogia SyncService";
        scheduler = Scheduler.Scheduler.Instance;
    }
    protected override void OnStart(string[] args) { scheduler.Start(); }
    protected override void OnStop() { scheduler.Stop(); }
    protected override void OnPause() { scheduler.Pause(); }
    protected override void OnContinue() { scheduler.Resume(); }
}
```

Figur 17. Kod för att skapa en Windows service.

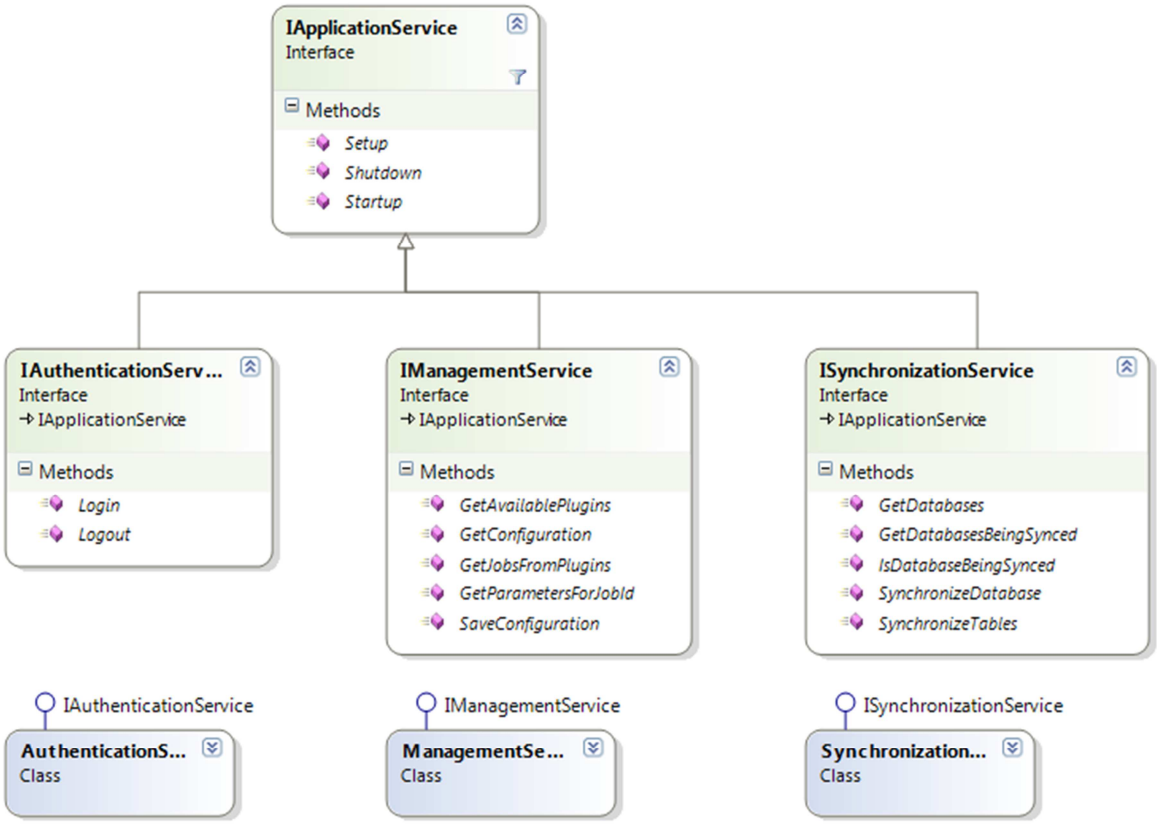
För att separera all funktionalitet i servicen från scheduleringen, synkningen samt kommunikationen med klienten så har det skapats en egen klass, kallad Scheduler, som innehåller all verklig funktionalitet. Scheduler-klassen har motsvarande funktioner som servicen, nämligen OnStart, OnStop, OnPause och OnContinue. Då en funktion anropas i servicen behöver den då bara anropa motsvarande funktion i Scheduler-klassen. Detta förenklar också testningen då man inte behöver installera om servicen i Windows för varje ändring, utan det går enkelt att skapa t.ex. ett litet program med en knapp för varje funktion och anropa Scheduler-klassen direkt. Scheduler-klassen är implementerad som en singleton, vilket betyder att då programmet körs kan det bara finnas en enda instans av Scheduler-klassen.

Scheduler-klassen har fyra huvudfunktioner som sker i följande ordning:

- Starta upp WCF-tjänsterna som klienten kommunicerar med.
- Läsa in konfigurationen för de olika jobb som blivit specificerade för schemaläggaren.
- Starta upp Quartz.NET-schemaläggaren.
- Läsa in konfigurationen och starta om Quartz.Net efter att konfigurationen uppdaterats.

3.3.2 Implementation av WCF-tjänster

Totalt implementeras tre WCF-tjänster enligt klassdiagrammet nedan:



Figur 18. Klassdiagram över SyncServicens WCF-tjänster.

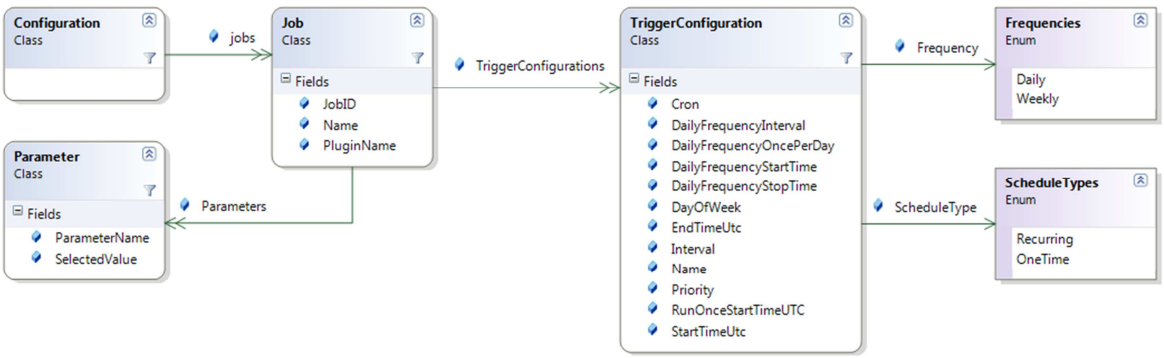
- Authentication Service – Denna tjänst har endast två funktioner Login och Logout. Dessa används för autentisering mellan klienten och servern så att endast en person med behörighet kan ändra konfigurationen av jobb och göra manuella synkroniseringar. Login-funktionen gör i sin tur ett anrop till BOOKIT-servern för att autentisera mot befintliga BOOKIT-användare. Detta betyder att för att kunna använda SyncService-verktygen måste användaren finnas uppsatt i BOOKIT.

- Management Service – Denna tjänst innehåller de funktioner som behövs för att kunna konfigurera, lägga till och ta bort jobb från schemaläggaren. Utöver att kunna spara och hämta konfigurationen finns även funktionalitet för att meddela alla övriga inloggade användare att konfigurationen ändrats så att en användare inte sparar över de övrigas ändringar.
- Synchronization Service – Denna tjänst innehåller funktioner för att hämta information om vilka databaser som synkroniseras för tillfället. Det finns även funktionalitet för att kunna manuellt synkronisera en alla eller enskilda tabeller i en databas som blivit uppsatt för synkroniseringsmodulen.

Dessa tjänster startas upp då Start-funktionen i Scheduler-klassen anropas och kommer sedan att köra tills Stop-funktionen anropas. För att den s.k. Callback funktionaliteten, som gör det möjligt för servern att anropa klienterna och inte bara andra vägen, ska fungera används TCP som kommunikationsprotokoll för tjänsterna.

3.3.3 Konfiguration

Konfigurationen av SyncService består av ett antal klasser. Huvudklassen, Configuration, har bara en lista av jobbklasser, Job, som finns uppsatta för schemaläggaren. Jobbklassen har en ID samt information om vilken dll jobbet ligger i och vad det heter. Ett jobb har dessutom en lista av trigger-konfigureringar och parametrar. Parameterklassen har enbart ett namn för parametern och ett värde. En trigger innehåller enbart information om när och inom vilka intervaller ett jobb ska köra. Nedan följer ett klassdiagram över Configuration-klassen och dess underklasser.



Figur 19. Klassdiagram över SyncServicens konfiguration.

Konfigurationen sparas ned som en xml fil på servern där den läses upp av SyncService-servicen då den startar. Konfigurationen kan skötas antingen via klienten för de som har behörighet eller genom att editera xml-filen för de som har tillgång till servern.

3.3.4 Schemaläggaren

Det som Quartz.NET definierar som ett jobb är egentligen bara en klass. De enda kraven för att en klass skall fungera som ett jobb i Quartz är att den implementerar interfacet IJob som finns definierat i Quartz och att klassens konstruktor inte har några argument. I IJob interfacet finns metoden Execute definierad och det är denna metod som körs när ett jobb startas av Quartz.

Ett jobb i Quartz kan skapas med följande kod:

```
public class MyJob : IJob
{
    // Konstruktorn får inte ta några parametrar.
    public MyJob() { }

    // Viktigt att implementera denna funktion!
    // Denna funktion är utgångsläget för ett jobb.
    public void Execute(JobExecutionContext context)
    {
        // Visa ett meddelande med nuvarande datum och tid
        MessageBox.Show(DateTime.Now.ToString());
    }
}
```

Figur 20. Implementation av ett Quartz-jobb.

För att lägga till jobbet ”MyJob”, som skapades ovan, till schemaläggaren kan följande kod användas (se figur 21):

```
private void CreateScheduler()
{
// Skapa ett nytt jobb med namnet JobName av typen MyJob.
JobDetail jobDetail = new JobDetail("JobName", typeof(MyJob));

// Skapa en trigger med namnet TriggerName som refererar till jobbet som
// skapades tidigare och som startar nu och kör igång jobbet en gång varje
// minut fram tills årsskiftet 2001/2012.
CronTrigger ct = new CronTrigger();
ct.CronExpressionString = "0 * * * * ? 2011";
ct.StartTimeUtc = DateTime.Now.ToUniversalTime();
ct.JobName = "JobName";
ct.Name = "TriggerName";

// Skapa schemaläggaren och starta den.
ISchedulerFactory schedulerFactory = new StdSchedulerFactory();
IScheduler scheduler = schedulerFactory.GetScheduler();
scheduler.Start();

// Lägg till jobbet och triggern.
scheduler.AddJob(jobDetail, true);
scheduler.ScheduleJob(ct);
}
```

Figur 21. Kod för att skapa en schemaläggare och lägga till ett jobb samt en trigger för jobbet.

Execute-metoden, som implementerades i MyJob-klassen, tar ett argument kallat JobExecutionContext, som också finns definierad i Quartz. Om man behöver passa parametrar till ett jobb då det körs går det bra genom att lägga till raden:

```
jobDetail.JobDataMap["Parameter1"] = "ParameterValue";
```

Figur 22. Kod för att lägga till en parameter till ett jobb.

och sedan läsa in den i jobbets Execute metod med:

```
String Parameter = context.JobDetail.JobDataMap["Parameter1"] as String;
```

Figur 23. Kod för att hämta en parameter i ett jobb.

För att underlätta för eventuella tillägg av synkroniseringsmöjligheter, som t.ex. andra typer av databaser eller filsynkronisering, skapades en abstrakt klass kallad AbstractJob. AbstractJob-klassen implementerar interfacet och Execute-funktionen. Klassen har utöver det två abstrakta funktioner som måste implementeras av de jobb som ärver denna basklass. Dessa två funktioner är GetParameters och ExecuteJob.

Funktionens GetParameters syfte är att då man från konfigurationen väljer att skapa ett jobb, returnera vilka parametrar detta jobb behöver och vilken eller vilka typer av värden de bör ha. ExecuteJob-funktionen är motsvarande funktion till ExecuteJob i de klasser som ärver Ijob. Det enda som sker i AbstractJobs Execute-funktion är att parametrarna läses ut och sparas i en variabel enligt koden i figur 23. Efter det anropas ExecuteJob-metoden som är implementerad i den egentliga jobbklassen.

För att ytterligare underlätta för tillägg och uppdateringar fasades AbstractJob samt jobbklasserna ut till sina egna projekt av typen klassbibliotek, d.v.s. dll-filer. Då schemaläggaren startas och läser in konfigurationen läser den också in den DLL-fil vars namn matchar ett visst mönster via reflection enligt koden i figur 24. Då denna DLL-fil lästs in till en assembly söks den rätta jobbklassen fram och laddas in i schemaläggaren. Detta gör att det går att separera olika typer av jobb i olika filer och enbart uppdatera en specifik fil om så behövs.

```
// PluginName är namnet på dll filen som jobbet finns i och
// JobName är namnet på jobbet (klassen)
private Type GetJobType(String PluginName, String JobName)
{
    String assemblyName = Constants.JobNamespace + PluginName + ".dll";
    String baselocation = AppDomain.CurrentDomain.BaseDirectory;

    Assembly assembly = Assembly.LoadFrom(baselocation + assemblyName);

    Type t = null;
    foreach (Type type in assembly.GetTypes())
    {
        if (type.Name == JobName)
        {
            t = type;
        }
    }
    return t;
}
```

Figur 24. Kod för att hämta ett jobb ur en DLL-fil.

3.3.5 Synkroniseringsmodulen

Synkroniseringsmodulen är ett jobb som baserar sig på AbstractJob-klassen som nämndes tidigare. Jobbets syfte är att läsa in de parametrar som blivit uppsatta för det jobbet och starta en synkronisering baserat på dessa.

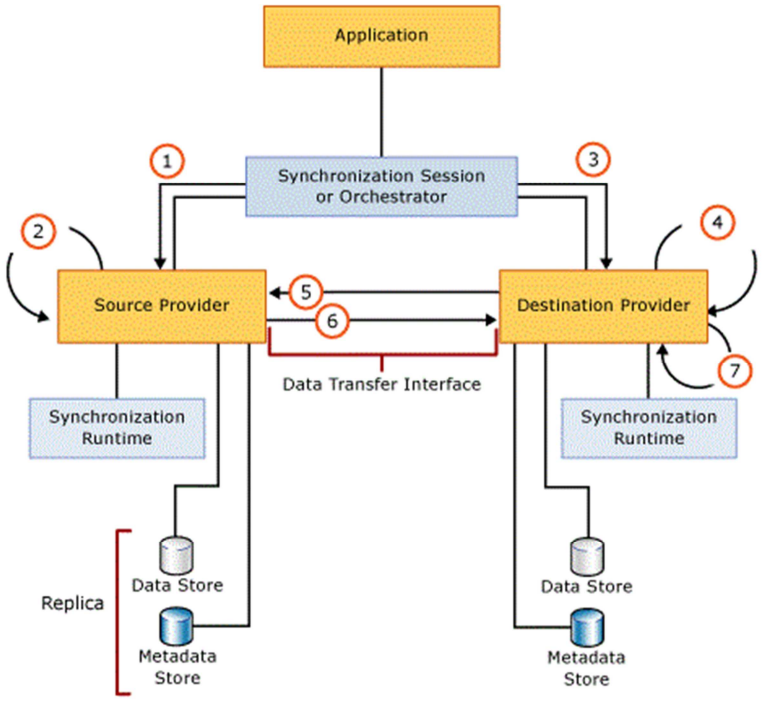
Synkroniseringsmodulens huvuduppgifter är:

- läsa upp konfigureringen för den databas som synkroniseringen gäller enligt de parametrar som passas till jobbet via schemaläggaren
- sköta loggning över statistik och eventuella felmeddelanden
- samt starta synkroniseringen.

All synkronisering av databasen sker sedan i synkroniseringsmotorn som startas via denna synkroniseringsmodul.

3.3.6 Synkroniseringsmotorn

Figuren nedan är baserad på en bild från Microsofts MSDN-sida (/2/) och visar huvudelementen i ett synkroniseringsscenario:



Figur 25. Schema över ett synkroniseringsscenario i Sync Framework

Beskrivning för de olika punkterna är:

1. Skicka destinationsdata till Source Provider.
2. Använd destinationsdata för att identifiera ändringar att skicka tillbaka.
3. Skicka källdata och metadata till Destination Provider.
4. Jämför data och ändringar samt meddela om eventuella konflikter.
5. Fråga efter objekt från källan.
6. Skicka objekt från källan.
7. Spara ner objekten.

3.3.6.1 Synkroniseringsmotorns uppbyggnad

Synkroniseringsmotorn ställer tre krav på de databastabeller som skall synkroniseras. Dessa är att kolumner för följande information skall finnas:

- en unik id för varje rad
- en tidsstämpel då raden blev skapad
- en tidsstämpel då raden blev senast uppdaterad.

Den unika id:n används för att matcha en rad mellan huvuddatabasen och den databas som synkroniseringen sker till. Tidsstämpeln då raden blev skapad används för att avgöra om raden har blivit skapad sedan den senaste synkroniseringen. Om så är fallet skapas ett insert-kommando för denna rad. Om tidsstämpeln för skapandedatum dock är äldre än senaste synkroniseringsdatum kontrolleras i stället uppdateringsdatumet. Om detta är efter senaste synkroniseringen skapas ett update-kommando. Om båda datumen är äldre än senaste synkroniseringsdatumet har denna rad inte ändrats och behöver alltså inte tas med i synkroniseringen.

För att även kunna synkronisera borttagna rader kommer en extra tabell att skapas. Varje gång man tar bort en rad ur en tabell i databasen kommer en rad att läggas till i denna tabell. Denna tabell innehåller tre kolumner:

- namnet på tabellen som en rad blivit borttagen ur
- en id för att kunna identifiera vilken rad som blivit borttagen
- och en tidsstämpel när raden blev borttagen.

Denna tabell kommer då att användas för att hitta matchande rad i de övriga databaserna som synkroniseras med denna.

Synchronization Orchestratorn från bilden är av typen SyncAgent i det här fallet. SyncAgentens enda uppgifter i synkroniseringsmotorn är att lagra information om de två olika Provider som används, namnet på tabellerna som skall synkroniseras och vilken typ av synkronisering som skall ske för enskilda tabeller (upload, download, bidirectional eller snapshot) samt, efter att alla övriga inställningar blivit gjorda, starta hela synkroniseringen.

Source Provider och Destination Provider från bilden ovan har som uppgift att hålla reda på den information som behövs för att kommunicera med databasen samt all information om vilken/vilka tabeller som skall synkroniseras. För att providern skall kunna lagra information om de olika tabellerna finns en SyncAdapter i providern.

SyncAdaptern har 8 st kommandon av typen IDbCommand som används i den här synkroniseringsmotorn, dessa är:

- SelectIncrementalInsertsCommand: Detta kommando används för att hämta den data som lagts till i databasen sedan den senaste synkroniseringen kördes.
- SelectIncrementalUpdatesCommand: Detta kommando används för att hämta den data som uppdaterats i databasen sedan den senaste synkroniseringen kördes.

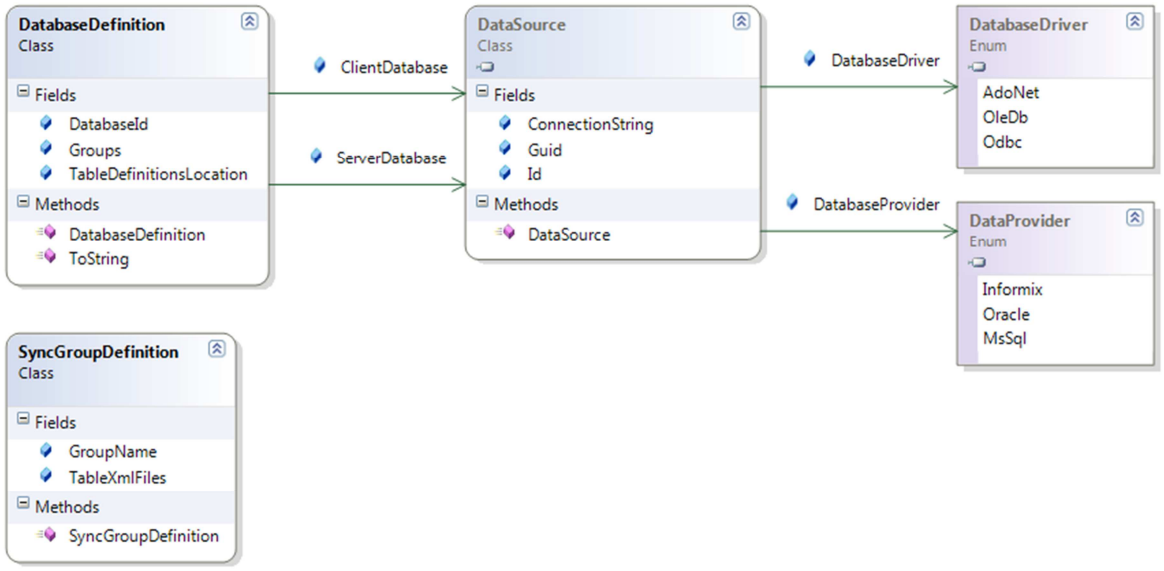
- SelectIncrementalDeletesCommand: Detta kommando används för att hämta information om det data som tagits bort ur databasen sedan den senaste synkroniseringen kördes.
- InsertCommand: Detta kommando sparar ner det data som hämtats upp via SelectIncrementalInsertsCommand
- UpdateCommand: Detta kommando uppdaterar det data som hämtats upp via SelectIncrementalUpdatesCommand
- DeleteCommand: Detta kommando tar bort en rad baserat på den information som hämtats upp av SelectIncrementalDeletesCommand
- SelectConflictDeletedRowsCommand: Detta kommando hämtar information om data där det uppstått konflikter då samma data tagits bort i flera databaser
- SelectConflictUpdatedRowsCommand: Detta kommando hämtar information om data där det uppstått konflikter då data t.ex. ändrats från två ställen

Dessa kommandon byggs upp dynamiskt från tabelldefinitioner som lagras i xml-format. Detta ger den möjligheten att enkelt lägga till eller ta bort databaser, tabeller eller även enskilda kolumner från synkroniseringen utan att behöva ändra något i programkoden.

Utöver de åtta nämnda kommandona finns ytterligare ett kommando som måste sättas på provider. Detta kommando är SelectNewAnchorCommand vars uppgift är att hämta det s.k. ankaret (anchor) som (i denna implementation) är den senaste raden som blivit uppdaterad i databasen. Detta ankare sparas sedan då synkroniseringen är färdig, så nästa gång synkroniseringen körs vet den från vilken rad i databasen den ska börja läsa och läser då fram till det nya ankaret.

3.3.6.2 Databasdefinitionen

Klassdiagrammet för databasdefinitionen ser ut på följande sätt:



Figur 26. Klassdiagram över en databasdefinition.

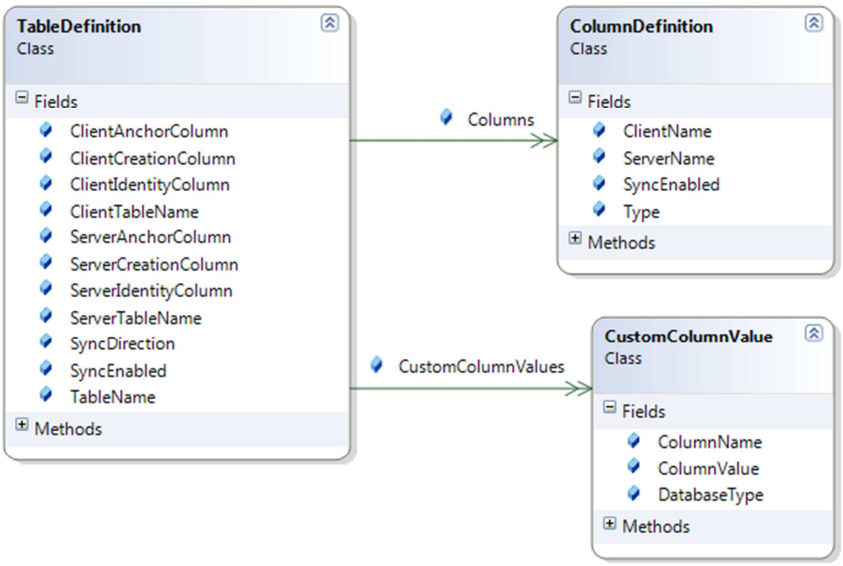
DatabaseId-fältet i DatabaseDefinition-klassen används för att identifiera vilken databas denna definition gäller. Detta fält används inte av synkroniseringsmotorn utan det är bara till för att användaren lättare skall kunna se vilken definition han konfigurerar. ClientDatabase och ServerDatabase är båda av typen DataSource och innehåller information som behövs för att kunna ansluta till respektive databas.

För att lättare kunna separera tabeller som inte är relaterade till varandra delas de in i olika grupper. Dessa grupper kan då synkroniseras skilt och på så sätt ge bättre prestanda då inte lika mycket arbetsminne går åt. I definitionen sparas information om dessa grupper i DatabaseDefinition-klassens Groups som är en lista av typen SyncGroupDefinition. SyncGroupDefinition innehåller ett namn på gruppen, enbart för att underlätta för användaren, och namnet på den xml-fil som innehåller tabelldefinitionen. Platsen var tabelldefinitionsfilerna finns är sparade i DatabaseDefinitions TableDefinitionsLocation.

DatabaseDriver anger vilken drivrutin som skall användas för att kommunicera med databasen och DatabaseProvider anger vilken typ av databas det är frågan om.

3.3.6.3 Tabelldefinitionen

Klassdiagrammet för tabelldefinitionen ser ut som följande:



Figur 27. Klassdiagram över en tabelldefinition.

I klassen TableDefinition finns motsvarande fält för de tidigare nämnda kraven på tabellerna (* står för Client samt Server):

- *IdentityColumn är namnet på den kolumn som innehåller den unika id för raderna i en tabell.
- *CreationColumn är namnet på den kolumn som innehåller tidsstämpel från när raden blev skapad.
- *AnchorColumn är namnet på den kolumn som innehåller tidsstämpel från när raden blev senast ändrad.

Förutom dessa sex fält finns även ClientTableName och ServerTableName som anger vad tabellen heter i de två databaserna. På grund av att databasstrukturen är lika i de två olika databaserna som kommer att synkroniseras, men namnet på tabellerna samt kolumnerna har blivit översatta till ett annat språk så behövs dessa fält för att kunna relatera tabellerna med varandra.

SyncDirection anger vilken väg data skall synkroniseras mellan databaserna, alternativen är:

- upload: Skicka ändrad data enbart från Client till Server.
- download: Skicka ändrad data enbart från Server till Client.

- bidirectional: Skicka ändrad data både från Client till Server och från Server till Client.
- snapshot: Skicka alltid all data från Server till Klient.

Fältet TableName används för identifiering av definitionen.

SyncEnabled är av en s.k. boolean-datatyp och kan antingen vara sant eller falskt (True/False), denna anger ifall denna tabell skall tas med i synkroniseringen eller inte.

Utöver dessa fält finns även två listor av andra objekt. Dessa är Columns som är en lista av ColumDefinition och CustomColumnValues som är en lista av CustomColumnValue.

Klassen ColumnDefinition innehåller namnet på en kolumn från både Server och Client, detta för att även kolumnerna kan vara översatta till kundernas egna språk. Type är den typ av fält denna kolumn innehåller, detta kan vara t.ex. datum, text, decimal eller heltal. SyncEnabled kan ha antingen värdet sant eller falskt och anger om denna kolumn skall synkroniseras eller inte.

Klassen CustomColumnValue används om man behöver spara extra information i den tabell som synkroniseras som inte finns i källdatan.

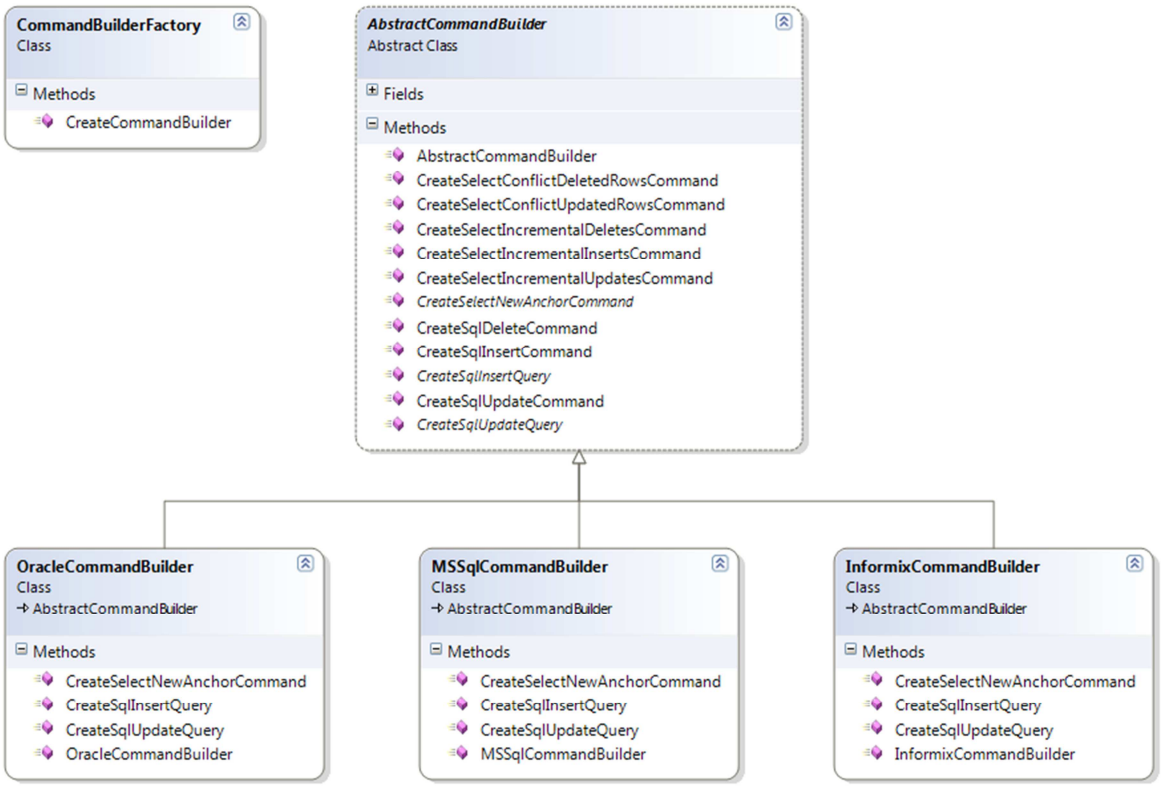
I figur 28 finns ett exempel på hur en xml-fil för synkroniseringsmotorns tabelldefinitioner ser ut:

```
<TableDefinition>
  <TableName>MyTable</TableName>
  <ServerTableName>MinTabell</ServerTableName>
  <ClientTableName> MyTable </ClientTableName>
  <ServerIdentityColumn>mt_serial</ServerIdentityColumn>
  <ClientIdentityColumn>Serial</ClientIdentityColumn>
  <ServerAnchorColumn>mt_stamp</ServerAnchorColumn>
  <ClientAnchorColumn>TimeStamp</ClientAnchorColumn>
  <ServerCreationColumn>mt_tekostamp</ServerCreationColumn>
  <ClientCreationColumn>CreationTimeStamp</ClientCreationColumn>
  <SyncEnabled>true</SyncEnabled>
  <SyncDirection>DownloadOnly</SyncDirection>
  <CustomColumnValues>
    <CustomColumnValue>
      <DatabaseType>Client</DatabaseType>
      <ColumnName>key</ColumnName>
      <ColumnValue>NEWID()</ColumnValue>
    </CustomColumnValue>
  </CustomColumnValues>
  <Columns>
    <ColumnDefinition>
      <ServerName>mt_serial</ServerName>
      <ClientName>Serial</ClientName>
      <Type>Int</Type>
    </ColumnDefinition>
    <ColumnDefinition>
      <ServerName>mt_stamp</ServerName>
      <ClientName>TimeStamp</ClientName>
      <Type>VarChar</Type>
    </ColumnDefinition>
    <ColumnDefinition>
      <ServerName>mt_tekostamp</ServerName>
      <ClientName>CreationTimeStamp</ClientName>
      <Type>VarChar</Type>
    </ColumnDefinition>
  </Columns>
</TableDefinition>
```

Figur 28. Exempel på en tabelldefinition i XML-format.

3.3.6.4 Dynamisk generering av SQL-kommandon

För att skapa kommandona för SyncAdaptrarna som används av ServerProvidern och ClientProvidern används en CommandBuilder-klass. På grund av att implementationen av SQL i de olika databaserna skiljer sig lite behövs i vissa fall databasspecifika implementationer av SQL-kommandon. Detta har lösts genom att skapa en abstrakt klass som de databasspecifika CommandBuilder-klasserna kan ärva och på så sätt inte behöva återimplementera den kod som går att återanvända mellan de olika databaserna.



Figur 29. Klassdiagram över SQL CommandBuilder-klasserna.

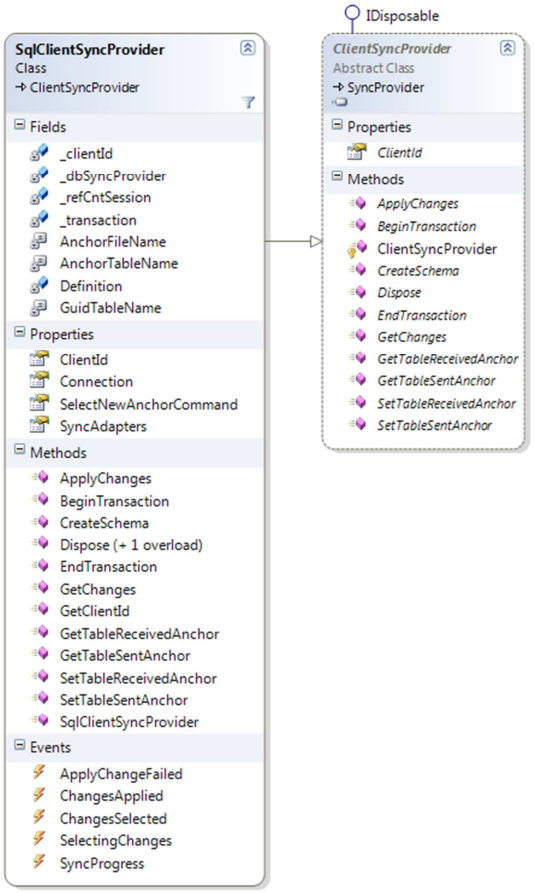
På grund av att de olika databaserna skiljer sig då man ger med en s.k. output-parameter med SQL-kommandot för att få tillbaka ett värde, har tre av funktionerna i AbstractCommandBuilder markerats som abstrakta, d.v.s. att de klasser som ärver denna klass måste implementera dem. I de databasspecifika CommandBuilder-klasserna finns då SQL-kommandona specificerade för att fungera med respektive databas.

För själva genereringen av SQL-kommandon läses först databas- och tabelldefinitionen in. CommandBuilderFactory skapar sedan en CommandBuilder för gällande databas utgående från vilken DatabaseProvider som angivits i databasdefinitionen. Efter att CommandBuildern skapats genereras de olika SQL-kommandona utifrån de tabelldefinitioner som satts upp för databasen.

3.3.6.5 Implementation av ClientSyncProvider-klassen

De två Provider-klasserna som används av SyncAgenten är alltid en av varje av antingen LocalProvider eller RemoteProvider. LocalProvider måste vara av typen ClientSyncProvider och RemoteProvider måste vara av typen ServerSyncProvider.

SyncFramework innehåller en färdig ServerSyncProvider för att kommunicera med databaser men om man vill använda en traditionell databas som LocalProvider måste man implementera en egen ClientSyncProvider för detta.



Figur 30. Klassdiagram över ClientSyncProvider-klassen.

En egen klass kallad SqlClientSyncProvider implementerades för att kunna använda SQL Server, Informix och Oracle som klientdatabaser. SqlClientSyncProvider ärver klassen ClientSyncProvider för att kunna användas som en LocalProvider för SyncAgenten. På grund av att alla funktioner i ClientSyncProvider är abstrakta måste de implementeras i den egna klassen.

För att underlätta implementationen av denna klass används en lokal DbServerSyncProvider i SqlClientSyncProvidern. Eftersom funktioner såsom ApplyChanges, BeginTransaction, EndTransaction, GetChanges samt Connection och SyncAdapters även finns hos DbServerSyncProvider kan dessa funktioner anropas via denna så att man inte behöver återimplementera hela denna funktionalitet.

Värt att notera dock är att DbServerSyncProvider ser på synkroniseringen från serverns perspektiv även om denna används i en klientens provider. Detta medför att man måste ändra riktningen på dataflödet i några av funktionerna. Ett av ställena där man måste ändra detta är i ApplyChanges. Innan man anropar den lokala DbServerSyncProviderns motsvarande funktion måste man ändra riktningen på synkroniseringen, alltså Upload →Download och tvärtom. Detta kan göras med koden i figur 31.

```
public override SyncContext ApplyChanges(SyncGroupMetadata groupMetadata
 System.Data.DataSet dataSet, SyncSession syncSession)
{
  foreach (SyncTableMetadata stm in groupMetadata.TablesMetadata)
  {
      if (stm.SyncDirection == SyncDirection.DownloadOnly)
        stm.SyncDirection = SyncDirection.UploadOnly;
      else if (stm.SyncDirection == SyncDirection.UploadOnly)
        stm.SyncDirection = SyncDirection.DownloadOnly;
  }

  SyncContext syncContext =
  _dbSyncProvider.ApplyChanges(groupMetadata, dataSet, syncSession);

  foreach (SyncTableMetadata stm in groupMetadata.TablesMetadata)
  {
    if (stm.SyncDirection == SyncDirection.DownloadOnly)
      stm.SyncDirection = SyncDirection.UploadOnly;
    else if (stm.SyncDirection == SyncDirection.UploadOnly)
      stm.SyncDirection = SyncDirection.DownloadOnly;
  }

  return syncContext;
}
```

Figur 31. Exempelkod för att ändra synkroniseringsriktningen i ClientSyncProviderns interna provider.

Det andra stället som måste ändras är i funktionen GetChanges där LastReceivedAnchor och LastSentAnchor måste svängas om. En Anchor är det som lagrar information om vilken rad som blev senast synkroniserad. Detta kan åtgärdas genom koden i figur 32.

```
public override SyncContext GetChanges
(SyncGroupMetadata groupMetadata, SyncSession syncSession)
{
  foreach (SyncTableMetadata metaTable in groupMetadata.TablesMetadata)
  {
    SyncAnchor temp = metaTable.LastReceivedAnchor;
    metaTable.LastReceivedAnchor = metaTable.LastSentAnchor;
    metaTable.LastSentAnchor = temp;
  }
  SyncContext context=_dbSyncProvider.GetChanges(groupMetadata, syncSession);
  foreach (SyncTableMetadata metaTable in groupMetadata.TablesMetadata)
  {
    SyncAnchor temp = metaTable.LastReceivedAnchor;
    metaTable.LastReceivedAnchor = metaTable.LastSentAnchor;
    metaTable.LastSentAnchor = temp;
  }
  return context;
}
```

Figur 32. På grund av ClientSyncProviderns interna Provider måste riktningen på ankaren ändras.

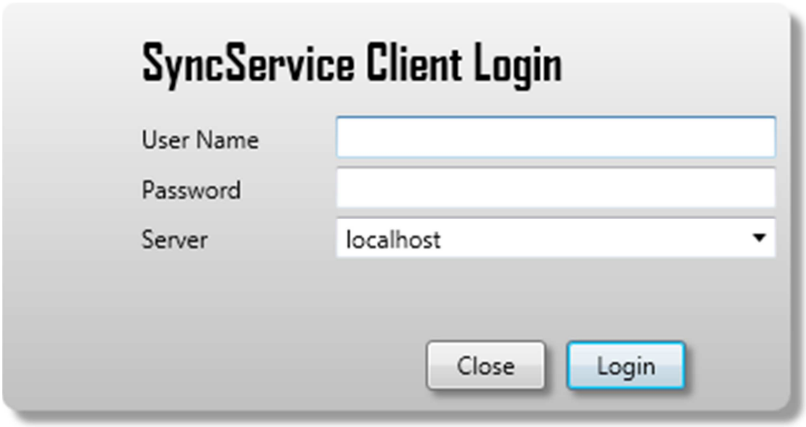
3.3.6.6 Batchning

Under utvecklingen av synkroniseringsmotorn då synkroniseringen testades uppstod ett problem under den första synkroniseringen då all data skall synkroniseras till en tom databas. På grund av de stora datamängderna tog arbetsminnet slut och programmet kraschade. Lösningen på detta problem var att implementera s.k. batchning. Detta innebär att all data inte synkroniseras på en gång utan i stället delas datan upp i olika segment som synkroniseras skilt för sig. Dessa segment delas upp och tas i den ordning som de satts in i databasen.

Eftersom Sync Framework har inbyggt stöd för batchning behövdes bara en parameter sättas för hur stora batchar den skall ta och kommandot för att hämta nya ankare behövdes uppdatera för att ta detta i beaktande så att det nya ankaret kan vara t.ex. 10 dagar från det förra ankaret. Detta löste problemet med att minnet tog slut i och med att mängden data per synkroniseringssession minskade.

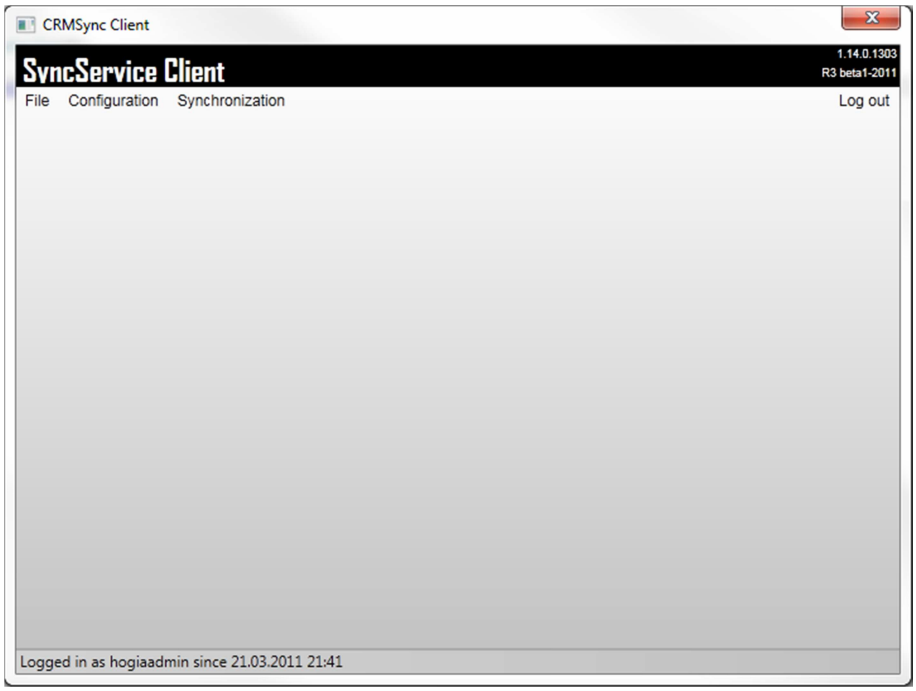
3.4 Klienten

För att kunna administrera synkroniseringstjänstens inställningar utvecklades en klient för detta bruk. Klienten är skriven i WPF och kommunicerar med synkroniseringstjänsten via WCF. För att kunna använda klienten måste man först autentisera sig. Eftersom detta program är utvecklat främst för att synkronisera BOOKIT:s databas så sker autentiseringen också mot denna databas. Detta betyder att för att kunna logga in i klienten måste användaren finnas registrerad som en BOOKIT-användare. I figur 33 följer en bild av inloggningsrutan som visas genast då man startar klienten.



Figur 33. Klientens inloggningsruta.

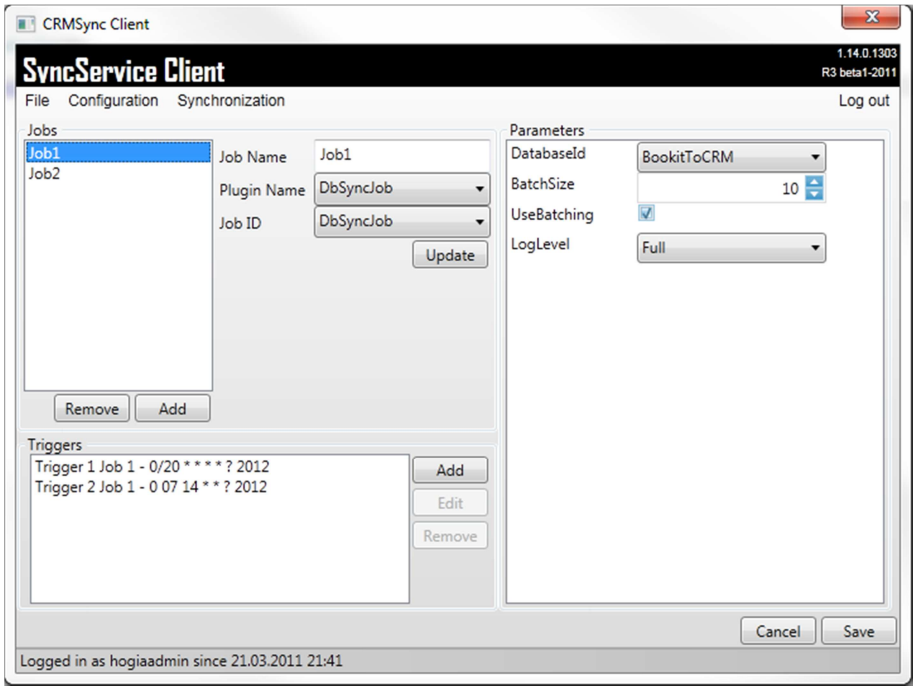
Efter en lyckad inloggning visas huvudfönstret. Huvudfönstret är byggt för att vara skalärbart, d.v.s. att man skall kunna lägga till ny funktionalitet om man t.ex. behöver en ny typ av synkronisering. Denna första version har dock bara funktionalitet för att konfigurera schemaläggaren och för att manuellt starta synkroniseringen av databaser. Stöd för administrering av tabeller och databaser ansågs inte vara nödvändigt i detta skede då definitionsfilerna kommer att uppdateras samtidigt som databasen uppdateras och detta program endast används för BOOKIT:s databas. I figur 34 följer en bild av huvudfönstret.



Figur 34. Klientens huvudfönster.

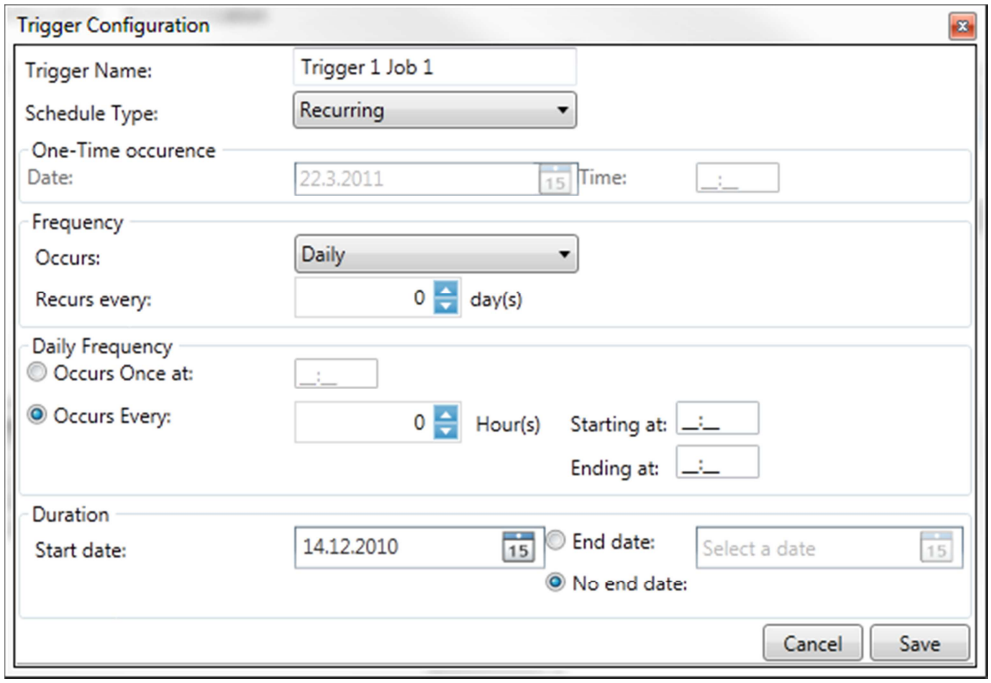
3.4.1 Konfigureringsmöjligheter

Schemaläggaren kan konfigureras via Configuration-alternativet i menyraden. Man kan där välja att lägga till/ta bort eller editera befintliga jobb för schemaläggaren.” Plugin name” är de plugin som finns att välja bland de dll-filer som finns på servern och som är definierade för att innehålla ett eller flera jobb.” Job ID” innehåller alla klasser som implementerar IJob interfacet som nämndes i kap. 3.3.4. Under Parameters-sektionen får man ställa in de parametrar som är definierade i jobbet. Triggers, som nämndes i kap. 2.6, kan läggas till/tas bort eller editeras under Triggers-sektionen. I figur 35 följer en bild av hur Configuration-sidan ser ut, där man lagt upp ett jobb för databassynkroniseringen plus ett annat jobb.



Figur 35. Jobbkonfigureringen

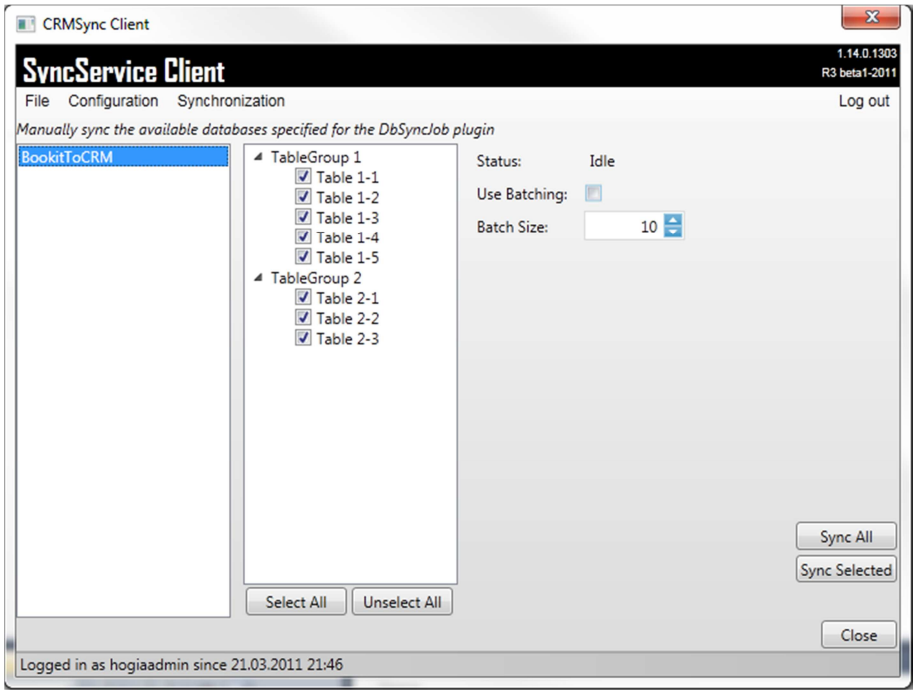
I dialogen nedan kan man skapa eller konfigurera en trigger. Det finns flera olika valmöjligheter och man kan även skapa flera triggrar för ett jobb ifall det inte går att lösa med en.



Figur 36. Dialog för att skapa en trigger.

3.4.2 Manuell synkronisering

Om man har behov av att synkronisera databasen utanför de specificerade tidsintervallen kan man även göra en manuell synkronisering. Genom att välja ”Synchronization” i menyn kommer man till den sida som visas i figur 37.



Figur 37. Manuell synkronisering.

Här kan man välja bland de databaser som finns definierade för databassynkroniserings-jobbet. Man kan även välja att synkronisera hela databasen eller valda tabeller samt om batchning skall användas.

3.5 Testning

Synkroniseringstjänsten har kontinuerligt testats i testmiljö under utvecklingen. När huvudutvecklingen är färdig kommer synkroniseringstjänsten att installeras på de testservrar som används av de kunder som visat intresse för detta. På dessa testservrar kommer den att synkronisera kundernas testdatabas till en skild kopia av den CRM-databas de använder. Extra loggning har lagts in i synkroniseringsjobbet under denna tid som, efter varje synkronisering, jämför data mellan båda databaserna och ser om det finns avvikelser.

4 Resultat

Resultatet av detta projekt i det här skedet är ett system med de nödvändiga komponenterna för schemaläggning och konfigurering av kontinuerlig databassynkronisering. De krav som ställdes på projektet har uppfyllts så att synkroniseringen nu är databasoberoende och körs på Windows Vista.

Under de tester som utförts under utveckling har detta program visat sig vara relativt effektivt samt pålitlig. Men eftersom tjänsten bara har testats i testmiljö i detta skede finns ingen konkret jämförelse med denna synkroniseringsmetod och den som användes tidigare.

Denna produkt är dock inte färdig utvecklad utan kommer att vidareutvecklas i ett senare skede. På server sidan är det främst synkroniseringsmotorn som kommer att vidareutvecklas. Stöd för SQL Servers inbyggda system för att integrera mot Sync Framework, kallat Change Tracking, blev inte implementerat i den första versionen, p.g.a. kravet på stöd för multipla databaser och då man ville ha en enhetlig lösning för alla databastyper. Om behov finns kommer detta också att implementeras i ett senare skede.

Även på klientsidan kommer det att ske en del vidareutveckling, bl.a. kommer bättre integrering att skapas för eventuella nya jobb på servern. Användargränssnittet kommer även att skapas i en version till, antingen som en skild applikation för webben, baserad på silverlight, eller som en del av BOOKIT-systemet.

5 Diskussion

5.1 Arbetets gång

Utvecklingen av detta projekt har framskridit ganska stadigt och även tidsmässigt har projektet gått ganska bra med tanke på att jag inte tidigare använt vare sig Sync Framework, Entity Framework 4 eller Quartz. Huvudorsaken som dragit ut på utvecklingstiden är att jag har varit i princip den enda utvecklaren av detta projekt och då har jag även haft andra projekt på sidan om.

Efter att valet hade fallit på Sync Framework påbörjades arbetet med att implementera en synkroniseringsmotor som använde detta. Arbetet med synkroniseringsmotorn påbörjades i mitten av november 2010 och första versionen tog ca två veckor att skapa.

Då första versionen av synkroniseringsmotorn var klar påbörjade jag arbetet med att implementera WCF-tjänsterna samt klienten och mitten av december påbörjades utvecklingen av schemaläggningen och dess konfiguration, vilket tog ca två veckor.

I januari 2011 fortsatte arbetet med att implementera databasoberoendet i synkroniseringsmotorn, tidigare hade den bara testats främst med SQL Server. Windows Servicen samt den manuella synkroniseringen implementerades ungefär samtidigt som detta. Efter att jag hade testat synkronisera större databaser märktes det att arbetsminnet inte räckte till i datorn så då påbörjades också arbetet med att implementera batchning av synkroniseringen. I början av februari började SyncService-projektet bli klart och loggning implementerades för att bättre kunna följa upp synkroniseringen under testningsfasen.

5.2 Tekniken

Att valet föll på Sync Framework drog ut på utvecklingstiden i början av utvecklingen bl.a. på grund av delvis bristfällig dokumentation, men visade sig efteråt vara ett bra val bl.a. på grund av den inbyggda funktionaliteten för batchning och synkronisering av ändrad data. Sync Framework skulle troligtvis ha varit ännu effektivare om jag hade kunnat använda mig av SQL Servers Change Tracking, men detta var tyvärr inte ett alternativ i detta projekt. Jag kan verkligen rekommendera Sync Framework som ett alternativ om man behöver ett system för att synkronisera data, speciellt om man utvecklar en klient som behöver fungera även i off-line läge.

Quartz.NET har visat sig fungera mycket bra i detta projekt och jag kan utan tvekan rekommendera Quartz.NET som schemaläggare. Det behövs relativt lite kod för att integrera den i det egna projektet och det är enkelt att implementera egna jobb samt triggers.

5.3 Kommentarer

I detta projekt hade ingen tidsram satts upp, utan det fick ta den tid det tog då jag även hade andra projekt. Men jag märkte dock tydligt att vid kommande projekt är det viktigt att kontrollera dokumentationen före projektplaneringen med tanke på hur den bristfälliga dokumentationen och/eller fungerande exempel på Sync Framework drog ut på utvecklingstiden i början av projektets gång, samt under implementationen av batchningen.

I övrigt är jag nöjd med valet av tekniker för detta projekt och slutligen kan jag konstatera att utvecklingen av detta projekt har varit väldigt intressant och lärorikt, speciellt eftersom jag har haft väldigt fria händer med planeringen, utvecklingen samt val av komponenter.

6 Källförteckning

/1/ 3.0 Summary – SubSonic :: Tome
SubSonic
http://subsonicproject.com/docs/3.0_Summary
(hämtat 12.03.2011)

/2/ Custom Provider Fundamentals
Microsoft MSDN
http://msdn.microsoft.com/en-us/library/bb902844%28v=SQL.110%29.aspx
(hämtat 20.03.2011)

/3/ IBM Informix
http://en.wikipedia.org/wiki/IBM_Informix
(hämtat 10.03.2011)

/4/ IBM Informix – Hassle-free data management
IBM
http://www-01.ibm.com/software/data/informix/discover-informix/hassle-free.html
(hämtat 10.03.2011)

/5/ McMurtry, Craig
Windows Communication Foundation Unleashed
2007, Sams Publishing
ISBN: 0-672-32948-4

/6/ Messaging Patterns in Service-Oriented Architecture Part 1
Microsoft MSDN
http://msdn.microsoft.com/en-us/library/aa480027.aspx
(hämtat 26.03.2011)

/7/ Microsoft Sync Framework
http://en.wikipedia.org/wiki/Microsoft_Sync_Framework
(hämtat 13.03.2011)

/8/ Microsoft Visual Studio Debugger
http://en.wikipedia.org/wiki/Microsoft_Visual_Studio_Debugger
(hämtat 23.01.2011)

/9/ Mindscape – LightSpeed
MindscapeHQ
http://www.mindscapehq.com/products/LightSpeed/default.aspx
(hämtat 12.03.2011)

/10/ Nathan, Adam
Windows Presentation Foundation Unleashed
2007, Sams Publishing
ISBN: 0-672-32891-7

/11/ .NET Framework
http://en.wikipedia.org/wiki/Microsoft_.NET#Microsoft_.NET
(hämtat 23.01.2011)

/12/ NHibernate
Wikipedia
http://en.wikipedia.org/wiki/NHibernate
(hämtat 12.03.2011)

/13/ Oracle Database
http://en.wikipedia.org/wiki/Oracle_Database
(hämtat 23.03.2011)

/14/ Quartz.NET - Enterprise Job Scheduler for .NET Platform
Sourceforge
http://quartznet.sourceforge.net/
hämtat 23.01.2011)

/15/ Visual C#
Microsoft msdn
http://msdn.microsoft.com/en-us/library/kx37x362.aspx
(hämtat 23.01.2011)

/16/ Visual Studio 2010 Professional – Microsoft Visual Studio
Microsoft
http://www.microsoft.com/visualstudio/en-us/products/2010-editions/professional
(hämtat: 23.01.2011)